Queen Mary
University of London

Parallelization solutions for the YNANO Discontinua Simulations in 2D

by

Fang Su

A thesis submitted in fulfilment of the requirements for the degree of Doctor of Philosophy and the Diploma of Queen Mary University Of London

School of Engineering and Materials Science

September 2015

For the bed of roses and the bed of nails, I dedicate this thesis

to myself .

Declaration of originality

The material presented in this thesis is entirely the result of my own independent research under the supervision of Professor Antonio Munjiza. All published or unpublished material used in this thesis has been given full acknowledgement.

Name: Fang Su Date: 4th May 2016

Signature:

List of Publications

Journal Papers:

Su, Fang, et al. "Use Improved Gradient Descent in Irregular Boundary Conditions in Molecular Dynamics." Applied Mechanics and Materials. Vol. 598. 2014.

Abstract

In the context of constant and fast progresses in nano technology, discontinua based computation simulations are becoming increasingly important, especially in the context of virtual experimentations. The efficiency of discontinua based nanoscale simulations are still limited by CPU capacity (the number of simulation particles in the system).

It is accepted that parallelization will play an important role in solving this problem. In this thesis, two parallelization approaches have been undertaken to parallelize the YNANO discontinua simulations. The scope of the work includes parallelization of the YNANO using the shared-memory approach OpenMP and the distributed-memory approach MPI, and also includes a novel MR_PB linear contact detection algorithm which can be used under periodic boundary conditions.

The developed MPI parallelization solutions are compatible with the MR linear contact detection algorithm used in the sequential YNANO, the developed solutions preserves the linearity of both MR_Sort and MR_Search algorithm.

The overall performance and scalability of the parallelization has been studied using nanoscale simulations in fluid dynamics and aerodynamics.

Acknowledgements

I would like to express my thanks to my supervisor Professor Ante Munjiza for his continuous support and help during my PhD studies. I would like to thank Dr. Tomas Lukas for his friendship and support during my difficult times (Thank you, Tomas), and Dr. Guillermo Gonzalo Shiava D'Albano for his worst-case-senario pushes during my less difficult times. To Dr. Mohammad Saadatfar for kindly allowing me access to their awesome HPC without which a major part of this thesis would not be possible. To my friend Weiqiang Zhang for his support over the years. To my dear mom and dad for their unwaivering steadfast love and moral support. To my evol (the future Tri-Dr.) Ahmad Amani for the chance encounter at the end of the journey. The rest of my acknowledgements go to my friends: Dr. Xiaolin Chen, Dr. Shirosh Tissera and (Doctor Doctor to be) Mouna Chetehouna, for their friendship and company.

Primo Victoria.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

SCOPE AND LAYOUT OF THE THESIS

1.1 Scope of the Thesis

Molecular Dynamics simulation (MD) was originially developed by Alder and Wainwright in the late 1950's and Rahman (independently) in the 1960's. A typical MD simulation consists of a number of atoms and molecules that are allowed to interact for a period of time. The interaction between the particles are defined by the interatomic potential or molecular force fields, and their trajectories are determined by numerically solving Newton's Second Law of Motion. MD was originally invented to solve problems within the field of theoretical physics, since then it has been applied in material science, chemical physics and biology over the years.[110, 105, 76, 77, 68]

A typical Molecular Dynamics simulation consists of discrete element based dynamics, contact search, contact interaction and motion integration. The most time consuming parts of the simulation are the contact search and the interaction, as the number of simulation particle increases by (millions),[147, 135, 110, 113, 122] more CPU time and computational power are required. Parallelization efforts for MD range from graphics processing unit (GPU) to clusters and desktop multiprocessor hardware.[164, 167, 147]

YNANO is an in-house MD code developed by Rougier and Munjinza, with a linear contact detection algorithm MR Sort and MR Search that achieve linear processing time to the number of simulation particles. If parallelization is employed, it is important to keep the linearity of the contact detection algorithm for the overall performance.

In this thesis, a shared-memory based parallelization solution and a novel distributed-memory based parallelization solution for Y-NANO 2D are presented. The developed parallelization solvers are described in detail, and the overall performance and scalability have been studied using numerical examples. The thesis also includes a novel

MR_PB linear contact detection algorithm which can be used in MD systems with periodic boundary conditions, the developed contact detection algorithm greatly expands the application of Y-NANO. Validations of the developed novel contact detection algorithm are also presented in the thesis.

1.2 Layout of the Thesis

This thesis is presented in the following way:

Chapter 2 presents a detailed introduction to YNANO Molecular Dynamics solutions, a short overview of various discontinua simulation methods in general, and an introduction to parallel processing. The introduction to YNANO focuses on the MR linear contact algorithm, a good understanding of which is important because it is the foundation of any further implementation upon the code. Parallel structures like Single Instruction Single Data (SISD), Single Instruction Multiple Data (SIMD) and multiple instructions as well as Multiple Data (MIMD) are discussed. Among which, MIMD system is of particular interest because the parallelization methods used in this thesis (both the shared-memory based parallelization solutions and the distributed-memory based parallelization solutions) fall into this category. Finally, evaluation metrics in parallel computing are presented, and as well as the ramifications of rounding error resulted from floating point arithmetic.

In chapter 3, a novel linear contact detection algorithm (MR_PB) is presented to implement periodic boundary conditions into YNANO. The developed algorithm is an implementation on the original MR contact detection algorithm. Both design and the implementations of the proposed new algorithm are explained in detail, including key issues such as how to preserve the cell space coherency within sorting, how to linearly parse the contact mask across the simulation domain, and etc. At the end of the chapter, test examples are carried out to validate the developed MR_PB algorithm, as well as an efficiency comparison analysis between MR_PB sort and binary sort.

Chapter 4 presents a novel approach to parallelize the Y-NANO with shared-memory based methods. The application program interface (API) used in this chapter is OpenMP. The developed parallelization solutions focuses on sharing the force calculation workload across multiple cores on a single node (with multiple cores). Both design and implementations of the proposed solution are explained in detail, including key issues like how to organize the search pointers for each processor, how work sharing concurrency is achieved, etc. Validations of the developed solution is performed in a $2D$ box filled with $250,000$ particles on 1, 4 and 8 cores on a single node.

Chapter 5 presents a novel approach to parallelize the Y-NANO on distributed-memory systems using MPI. A spatial decomposition strategy and memory management system that piggy-backs on the existing linear MR contact detection algorithm are presented. Details of the design and implementation are discussed in detail, including key issues such as the buffer zone scheme, communication, interfacial particle force transmission, particle migration, and linear sorting of the local particle and immigrant particles.

Chapter 6 presents the verification and performance tests of the proposed MPI solutions of Y-NANO on the distributed-memory systems. The developed parallelized code is tested on a $2D$ box filled with $250,000$ gaseous Argon particles with random initial velocity on 1, 4, 32 and 64 cores. The validity of the solutions is confirmed by comparing the general trend in motion of the simulation particles as well as the evolution of the system's total kinetic energy between a sequential version of the code and a parallelized version of the code.

Chapter 7 presents a numerical example in the field of fluid dynamics in which performance and scalability of the developed code are further studied and analysed. The results show good agreement with the theoretical results.

Finally, Chapter 8 provides a summary of the whole thesis, a conclusion. The chapter also discusses further possibilities of future work in the field.

Chapter 2

Introduction to Existing Domain Decomposition Based Parallelization Methods of Discontinua

2.1 Introduction

The performance improvements of microprocessor has led to massive development in the field of Computational Mechanics. However as the size of single processor drops to atomic scale, it proves difficult to further decrease the size without considering the quantum effects, therefore in recent years, the manufactures have started to build multicore chips to keep improving the CPU performance.

To fully utilize the vast possibilities that come with multicore processors, a sequential code needs to be parallelized.[159] It can be reasonably expected that parallel architectures like a High Performance Computing (HPC) cluster will play a huge role in the future.[34, 40, 41]

The rest of the chapter provides a short overview of the discontinua methods, parallel architectures as well as parallelization strategies used in MD.

2.2 Computational Methods of Discontinua

2.2.1 Introduction

In solving many engineering problems, the medium or the material could be considered continuous. For example, the Finite Element Method (FEM) is developed upon this assumption. It discretizes the continuum by dividing it into small parts called finite elements of different shapes and orders.[136]

However, there is a wider range of problems which cannot be solved using the continuum assumption. For example in a gaseous system, when the characteristic length of the simulation object is comparable to the characteristic length of the system, the continuum assumption fails to provide a realistic representation of the physical system. For problems like this, the assumption of discontinuum needs to be employed and methods of discontinua have been developed to solve them.

Methods of discontinua include:

- Discrete Element Method (DEM)
- Direct Simulation of Monte Carlo (DSMC)
- Molecular Dynamics (MD)

Discrete Element Method. DEM is developed by Cundall in 1971 to solve rock mechanics problems. The method is built upon solving equations of motion for each simulation particle separately by calculating the inter-particle forces from the inter-particle interactions. Further reading on the method can be found in a handful of books.[37, 169] Particularly, Discrete Element Method coupled with Finite Element Method (FDEM) has been used to studied an extensive range of systems and problems, e.g. geomechanical problems,[117, 78] block caving[93, 45] and a wide range of mechanical problems.[29]

Direct Simulation of Monte Carlo. DSMC is a probabilistic molecular model developed by Bird in the late 1960's, it is originally developed to simulate rarefied gas flow in high altitudes. DSMC employs simulated molecules to represent a fixed number of real molecules, thus greatly reduces the simulation size. DSMC also uses a statistical collision detection model between particles. The governing equation for DSMC is the Boltzmann equation, and therefore DSMC is strictly applied only to dilute gas flows.[17, 132, 146]

2.2.2 The Molecular Dynamics Simulations Methods

Unlike DSMC, Molecular Dynamics (MD) is a deterministic computational method that studies a system's time-dependent behaviour. In a typical MD system, all particles are governed by Newton's second law of motion,[55,99]

$$\mathbf{a} = \frac{\mathbf{f}}{m} \tag{2.1}$$

in which, $\mathbf{f}$ is the external force acted on a given particle, m is the mass of the particle, and $\mathbf{a}$ is the acceleration of the particle in response to the external force.

A typical MD system has three main steps:

- Initial configuration: Set up an MD simulation with initial conditions: particle velocity and geometry. Particle position and velocity information are crucial in determining the trajectory of each particle in the system.
- Contact detection: At each time step and for every particular particle i in the system, a contact detection algorithm is called to search around it within a cut-off vicinity for all particles that are considered close enough, beause the potential field between the conatct pairs will alter the pair's trajectory for the next timestep. Contact interaction: For every particle pair, the inter-particle distance could be readily obtained by comparing the coordinates, and subsequently the interaction force could be calculated using the chosen potential force field function. By summing up all the external interaction forces exerted on particle i, the particle's acceleration as a result to external forces could be calculated. This step is repeated on every particle in the system.
- Time integration of motion equation: every particles' acceleration information will then be used to update the particle velocity and position to be used in the next time step.

With Newton's second law of motion and by discretization on the time domain, MD computes the dynamic trajectory of a system comprising of a large number of particles. MD was developed by Alder and Wainwright in late 1950's and Rahman (independently) in the 1960's. Since then, a number of detailed implementations on specific steps have been developed, such as the Position Verlet (PV), Forest&Ruth and etc., time integration scheme, and a series of solutions on contact detection like bin sort and bubble sort etc. In this study, the PV time integration scheme is used, and the MR contact detection algorithm is used in the contact sorting and detection process.

2.2.2.1 The Theoretical background

The aim of computer simulations is to reproduce the reality as accurately as possible. It in essence works like a function: the function (algorithm model) performs some transformation on the inputs (key variables) and then an output (measurement result) is produced. How to specify the function is therefore highly crucial to the validity of the whole procedure. In a nutshell, a proper mathematics model is the key to a computational simulation. The model is in essence a set of governing equations.[136, 140, 55, 99]

There are two different models to simulate a given material: the continua approach (macroscopic) and the discontinua approach (microscopic).

The discontinua or discontinuum approach views the material as a collection of particles, and the macroscopic properties are merely the emergent properties of the interaction among constituting parts whose motions are described by the Newton's equations of motion. In essence, the discontinua approach sets out to obtain the macroscopic properties of a system by deriving from the microscopic properties of the constituting elements in a system. However in the continuum view, the material is viewed as a whole and is considered infinitely dividable. Therefore, the molecular details of the material are simply redundant and can be wholesomely replaced by the macroscopic properties.

It should be pointed out that different models are representations of the same physics law under different experiment conditions, because under different conditions, major contributing factors weigh differently, threrefore some kind of generalizations can be made to avoid redundancy.

In a gaseous of fluid system, the fine line between the two different views lies where the characteristic length l of the system is comparable to the mean free path λ of the constituting particles. Below the line, the continua approach will no longer produce truthful results, and this is when the discontinua approach must be adopted. The free mean path λ is given by:

$$\lambda = \frac{kT}{\sqrt{2}\pi\sigma^2 p} \tag{2.2}$$

in which, k is the Boltzmann constant, T is the system temperature, p is the system pressure, and σ is the gas molecular diameter.

The continua approach works well in system with a Knudsen number (Kn) smaller than 0.1 ($Kn < 0.1$) . The Knudsen number describes the rarefaction of a system and is given by:

$$Kn = \frac{\lambda}{l} \tag{2.3}$$

The Kinetic Theory The Kinetic Theory can be used to describe the discontinua approach, and the Navier-Stokes equations can be used to describe the continuum approach. In this study, only the Kinetic Theory will be introduced. The aim of the Kinetic Theory is to explain how macroscopic variables, e.g. system pressure, temperature etc., can be obtained from microscopic gas particle motion.

The Liouville equation uses the gas particles' position and velocity information to describe a system, it is the basic statistical equation that describes gas behaviour, given by

$$\frac{\partial P}{\partial t} + \sum [\mathbf{v}_i \frac{\partial P}{\partial \mathbf{r}_i} + \mathbf{f}_i \frac{\partial P}{\partial \mathbf{v}_i}] = 0 \tag{2.4}$$

where, $P = P(r, v, t)$ is the probability of finding a molecule within a differential phase space volume, N is the total number of particles in the system, v_i and r_i are the velocity and position of particle i at time t respectively, $\mathbf{f}_i$ is the net external force exerted on the particle. With the help of Bogoliubov–Born–Green–Kirkwood–Yvon hierarchy (BBGKY hierarchy), the Liouville equation eq.2.4 can be integrated repeatedly to bear a resemblance to the Boltzmann equation for monatomic gas, given by

$$\frac{\partial (nf)}{\partial t} + \mathbf{v} \cdot \frac{\partial (nf)}{\partial \mathbf{r}} + \mathbf{f} \cdot \frac{\partial (nf)}{\partial \mathbf{v}} = \frac{\partial_{collision}(nf)}{\partial t} \tag{2.5}$$

where, n is the number density of gas, f is the normalized distribution function. The left hand side of the equation represents the changes of the molecules in position space and velocity space. The second term addresses the particle convection in position space $d\mathbf{r}$ due to velocity $\mathbf{v}$, the third term addresses the influence of external force $\mathbf{f}$, which leads to convection of particles in velocity space $d\mathbf{v}$. The right hand side of the equation $\frac{\partial_{collision}(nf)}{\partial t}$ is the collision term, it describes the change in the number of particles due to particle-particle collision. For a dilute gas system, it is reasonable to assume that all particle collision are binary collisions, that is, the collision happens between two particles only.

For a gas particle moving in three-dimensional space, the phase space in Boltzmann equation is a six-dimensional space and can be split into two subspaces: the position

space r and the velocity space v.

$$\begin{bmatrix} x \\ y \\ z \\ v_x \\ v_y \\ v_z \end{bmatrix} \tag{2.6}$$

For a given physics quantity A, the Boltzmann equation eq.2.5 could be integrated over the velocity space as

$$\int_{-\infty}^{\infty} \frac{\partial(nAf)}{\partial t} d\mathbf{v} + \int_{-\infty}^{\infty} A\mathbf{v} \cdot \frac{\partial(nf)}{\partial \mathbf{r}} d\mathbf{v} + \int_{-\infty}^{\infty} A\mathbf{f} \cdot \frac{\partial(nf)}{\partial \mathbf{v}} d\mathbf{v} =$$

$$\frac{1}{2} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \int_{0}^{4\pi} n^2 (A^* + A_1^* - A - A_1) f_1 f \, v_r \sigma d\Omega d\mathbf{v} d\mathbf{v}_1 \tag{2.7}$$

in which, $A^* + A_1^* - A - A_1$ denotes how quantity A changes when collision type $v, v_1 \rightarrow v^*, v_1^*$ happens. The first term on the left side can be written as

$$\int_{-\infty}^{\infty} \frac{\partial(nAf)}{\partial t} d\mathbf{v} = \frac{\partial}{\partial t} \int_{-\infty}^{\infty} nAf d\mathbf{v} = \frac{\partial}{\partial t}(n\overline{A}) \tag{2.8}$$

The second term on the left side can be written as

$$\int_{-\infty}^{\infty} A\mathbf{v} \cdot \frac{\partial(nf)}{\partial \mathbf{r}} d\mathbf{v} = \int_{-\infty}^{\infty} \nabla \cdot (n\mathbf{v}Af) d\mathbf{v} = \nabla \cdot (n\overline{\mathbf{v}A}) \tag{2.9}$$

The third term on the left side can be written as

$$\int_{-\infty}^{\infty} A\mathbf{f} \cdot \frac{\partial(nf)}{\partial \mathbf{v}} d\mathbf{v} = \int_{-\infty}^{\infty} \mathbf{f} \cdot \frac{\partial(nAf)}{\partial \mathbf{v}} d\mathbf{v} - \int_{-\infty}^{\infty} \mathbf{f} \frac{\partial A}{\partial \mathbf{v}} nf d\mathbf{v} = -n\mathbf{f} \cdot \frac{\partial \overline{A}}{\partial \mathbf{v}} \tag{2.10}$$

Therefore eq.2.7 can be rewritten as

$$\frac{\partial(n\overline{A})}{\partial t} + \nabla \cdot (n\overline{\mathbf{v}A}) - n\mathbf{f} \cdot \frac{\partial \overline{A}}{\partial v} = \triangle[\overline{A}] \tag{2.11}$$

jioklIn the cases where a the quantity A is chosen as the mass m, the momentum $m\mathbf{v}$ or the kinetic energy $1/2mv^2$, the collision term on the right is evaluated to zero, due to

conservation principles.

$$\frac{1}{2}\int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\int_{0}^{4\pi} n^2(A^*+A_1^*-A-A_1)f_1 f\, v_r \sigma d\Omega d\mathbf{v} d\mathbf{v}_1 = 0 \tag{2.12}$$

Thus, three fundamental conservation laws can be obtained, namely the conservation of mass, conservation of momentum, and the conservation of energy.

Conservation of Mass In eq.2.11, substitute A with particle mass m,

$$\frac{\partial(nm)}{\partial t} + \nabla\cdot(nm\overline{\mathbf{v}}) = 0 \tag{2.13}$$

$$\frac{\partial\rho}{\partial t} + \nabla\cdot(\rho\mathbf{v_s}) = 0 \tag{2.14}$$

in which, $\rho = nm$ is the fluid density, and $\mathbf{v}_s = \overline{\mathbf{v}}$ is the stream velocity. Noted that the velocity of a particle in a stream consists of two parts: the stream velocity $\mathbf{v_s}$ and the thermal velocity $\mathbf{v_t}$:

$$\mathbf{v} = \mathbf{v_s} + \mathbf{v_t} \tag{2.15}$$

With the concept of substantial derivative:

$$\frac{D}{Dt} = \frac{\partial}{\partial t} + \mathbf{v_s}\cdot\nabla \tag{2.16}$$

eq.2.13 can be transformed into:

$$\frac{D\rho}{Dt} + \rho(\nabla\cdot\mathbf{v_s}) = 0 \tag{2.17}$$

eq.2.17 is the continuity equation or conservation of mass equation.

Conservation of Momentum Conservation of momentum. In eq.2.7, substitute A with particle momentum $m\mathbf{v}$,

$$\frac{\partial(n\overline{m\mathbf{v}})}{\partial t} + \nabla\cdot(n\overline{\mathbf{v}m\mathbf{v}}) - \rho\mathbf{f} = 0 \tag{2.18}$$

since $\rho = nm$, eq.2.18 is transformed into:

$$\frac{\partial(\rho\overline{\mathbf{v}})}{\partial t} + \nabla\cdot(\rho\overline{\mathbf{v}\mathbf{v}}) - \rho\mathbf{f} = 0 \tag{2.19}$$

After some transformation, it arrives at

$$\rho \frac{D\mathbf{v_s}}{Dt} + \nabla p - \nabla \cdot \tau - \rho \mathbf{f} = 0 \tag{2.20}$$

where, p is the scalar pressure given by

$$p = \frac{1}{3}\rho(\overline{v_{tx}^2} + \overline{v_{ty}^2} + \overline{v_{tz}^2}) = \frac{1}{3}\rho\overline{v_t^2} \tag{2.21}$$

and τ is the viscous stress tensor given by eq.2.22

$$\tau = -\mathbf{p} + \delta p \tag{2.22}$$

Conservation of Energy In equation eq.2.7, substitute A with particle energy $\frac{1}{2}mv^2$,

$$\rho \frac{\partial\left(\frac{\rho\overline{v^2}}{2}\right)}{\partial t} + \nabla \cdot (\frac{\rho\overline{\mathbf{v}v^2}}{2}) - \rho \mathbf{f} \cdot \mathbf{v_s} = 0 \tag{2.23}$$

The final representation of the conservation energy equation is given at:

$$\rho \frac{D\left(\frac{\overline{v_t^2}}{2}\right)}{Dt} = -\nabla \mathbf{q} + \Phi - p(\nabla \cdot \mathbf{v_s}) \tag{2.24}$$

in which, $\mathbf{q}$ is the heat flux vector given by eq.2.25, Φ is the viscous dissipation function defined by eq.2.26

$$\mathbf{q} = \frac{1}{2}\rho \mathbf{v_t^2} \mathbf{v_t} \tag{2.25}$$

$$\Phi = (\tau \cdot \nabla) \cdot \mathbf{v_s} \tag{2.26}$$

2.2.2.2 Initial Configuration

It has been demonstrated that gaseous atoms in a closed system will eventually relax towards the Maxwellian velocity distribution, regardless of the initial velocity field that is assigned to it. Therefore, the sensible thing to do is to assign an initial velocity field that conforms to the Maxwellian distribution function, with reference to the desired temperature.

Given that the two components of velocity are independent from each other, the Maxwellian distribution could be expressed as

$$f_M(v_{tx}) = \sqrt{\left(\frac{m}{2\pi kT}\right)} exp(-\frac{mv_{tx}^2}{2kT}) \tag{2.27}$$

in which, m is the atoms mass, k is the Boltzman constant, and T is the thermodynamic gas temperature.

For $2D$ cases, system temperature evaluates to

$$T = \frac{2}{3}\frac{m\overline{v^2}}{2k} = \frac{2}{3}\frac{\overline{E_k}}{k} \tag{2.28}$$

Furthermore, each component of the molecular velocity in the Maxwellian Distribution function can also be described in a Gaussian probability function, given by:

$$P_g(\mu, \sigma^2) = \frac{1}{\sigma\sqrt{2\pi}} exp(-\frac{(\xi - \mu)^2}{2\sigma}) \tag{2.29}$$

in which, σ is the standard deviation and μ is the mean. The Gaussian probability function will be first used to decide the velocity distribution of the atoms in the system, and then all atoms will be properly scaled according to the desired initial temperature.

After attributing a Gaussian distribution of velocity with reference to the system temperature, it is necessary to perform a procedure of linear momentum equilibration and angular momentum equilibration. This is an essential benchmarking step because for a system in equilibrium, its total particle momentum and angular momentum should be equal to zero.

Linear momentum equilibration.

First of all, the net momentum of the system $\mathbf{P}$ is calculated.

$$\mathbf{P} = \sum_{i=1}^{N} (m_i \mathbf{v}_i) \tag{2.30}$$

in which N is the total number of particles within the system, m_i is the particle mass, and $v_i$is the velocity of the particle.

The net momentum $\mathbf{P}$ of a closed system should be zero, if $\mathbf{P}$ doesn't evaluate to zero then further scaling is needed to perform on each particles according to the linear momentum equilibrium criteria as follows

$$v_i^{new} = v_i^{old} - \frac{\mathbf{P}}{N \times m_i} \tag{2.31}$$

in which v_i^{old} and v_i^{new}} are the velocity of particle i before and after the update respectively.

Angular momentum equilibration. The mechanism of angular momentum equilibration is similar to that of the linear momentum equilibration. The net angular velocity could be obtained by

$$\omega = \mathbf{I}^{-1}\mathbf{L} \tag{2.32}$$

in which, ω is the net angular velocity of the system, $\mathbf{I}$ is the system inertia tensor, and $\mathbf{L}$ is the angular momentum. $\mathbf{I}$ is given by

$$\mathbf{I} = \begin{bmatrix} I_{xx} & I_{xy} & I_{xz} \\ I_{yx} & I_{yy} & I_{yz} \\ I_{zx} & I_{zy} & I_{zz} \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{N} m_i(y_{ri}^2 + z_{ri}^2) & -\sum_{i=1}^{N} m_i x_{ri} y_{ri} & -\sum_{i=1}^{N} m_i x_{ri} z_{ri} \\ -\sum_{i=1}^{N} m_i x_{ri} y_{ri} & \sum_{i=1}^{N} m_i(x_{ri}^2 + z_{ri}^2) & -\sum_{i=1}^{N} m_i y_{ri} z_{ri} \\ -\sum_{i=1}^{N} m_i x_{ri} z_{ri} & -\sum_{i=1}^{N} m_i y_{ri} z_{ri} & \sum_{i=1}^{N} m_i(x_{ri}^2 + y_{ri}^2) \end{bmatrix} \tag{2.33}$$

The angular momentum $\mathbf{L}$ is given by

$$\begin{aligned} L &= \sum_{i=1}^{N} \mathbf{r}_{\mathrm{ri}} \times \mathbf{p}_{\mathrm{i}} \\ \mathbf{r}_{\mathrm{ri}} &= \begin{bmatrix} \mathrm{x_{ri}} \\ \mathrm{y_{ri}} \\ \mathrm{z_{ri}} \end{bmatrix} = r_{\mathrm{i}} - r_{\mathrm{cm}} \end{aligned} \tag{2.34}$$

in which, $\mathrm{x_{ri}}$, $\mathrm{y_{ri}}$ and $\mathrm{z_{ri}}$ are the relative coordinates of the particle with reference to the centre of mass r_{cm} of the system, $\mathbf{p}_i$ is the linear momentum of the particle. r_{cm} could be obtained by

$$r_{\mathrm{cm}} = \frac{\sum_{i=1}^{N} m_i \, \mathbf{r}_{\mathrm{i}}}{\sum_{i=1}^{N} m_i} \tag{2.35}$$

Now, the particle's velocity field is rescaled with reference to angular momentum as

$$\mathbf{v}_i^{new} = \mathbf{v}_i^{old} - \omega \times \mathbf{r}_{\mathrm{i}} \tag{2.36}$$

2.2.2.3 Contact Detection

In MD, particles are able to move around freely and some of them may come into contact with each other. It is therefore particularly important to develop an efficient

algorithm to address the contact detection issue for systems comprising of hundreds of thousands of particles. It is important to keep tracks of atoms because whether two particles are in contact or not is decided by the distance between these two particles.

For a given particle, the most straight-forward solution is the Direct Check which calculates the minimum distance between the particles against every particle pair and decide if each pair is close enough to be in contact. The Direct Check is very time-consuming and computationally-inefficient, for N particles the number of check times on each possible particle pair is given by

$$T \propto N \log_2 N \tag{2.37}$$

A more efficient way for contact detection is to grid the whole particle domain (the grid size is the cut-off radius), and then map all particles to the grid boxes (bounding boxes). Therefore, for a given particle, it can only be in contact with the particles in its surrounding bounding boxes. Thus, instead of wasting a huge amount of time in calculating the distance between every particle pair, the whole process can be speeded up by performing a preliminary bounding box filtering, followed by calculating the distance between the particle pairs of interest.

As is evident from the above analysis that repetition exists in the repeated filtering procedure of every particle repeated for N times, if the spatial information can be shared among subsequent procedures, the efficiency can thus be greatly boosted. One such solution is to transform the cell list into a sorted cell list which preserves the spatial information of the particle system.

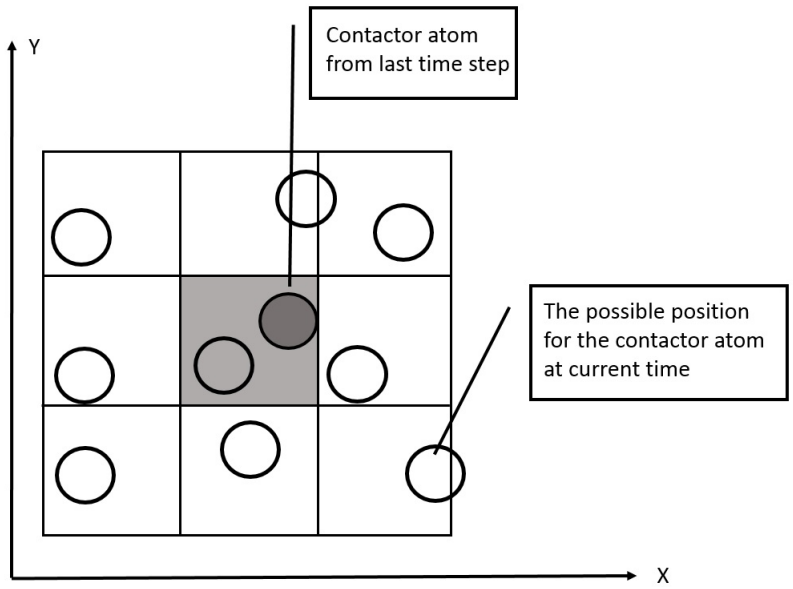

Figure 2.1: The possible positions for the contactor atoms at current time.

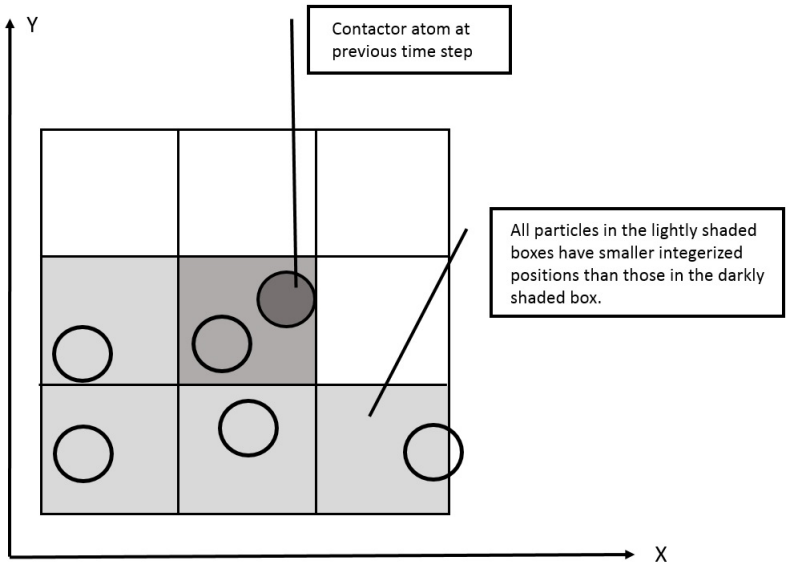

Figure 2.2: When the current atom position is smaller than the previous position, the nearly-sorted list needs to be resorted.

One obvious way to sort a list is using the binary sort, it has a processing time proportional to N^2. A number of contact detection algorithms have been developed to address the contact detection problem more efficiently in specific problem settings, for instance Position Code Algorithm, MR and various other contact detection algorithms. The vast majority of the newly devised contact detection algorithms can be roughly

divided into two categories: space-based search and body-based search.

The MR sorting agorithm offers linear processing time which is proportional to N, because it sorts on a nearly sorted list. The basic assumption of the MR-sort is that a particle cannot move farther than the grid size in a single time step. This is also called the stability criterion.

The MR contact algorithm is a linear complexity algorithm, thus, for a system with a total number of N particles, the total detection time T is proportional to N. The advantage of this algorithm is obvious when it is used to handle systems comprising of millions or billions of particles.

$$T \propto N \tag{2.38}$$

The MR contact detection algorithm has two parts, a so called MR_sort and followed by a so called MR_search.

The difficulty in sorting a nearly sorted list is that the non-directional movement of the particles might mess up the ordering in a massive scale. The MR-sort solves this problem by breaking up the movement of an atom into (for a $2D$ scenario) two basic directional classes: horizontal and vertical, and it further breaks down the two classes into a cross combination of right/left with up/down. MR_sort simultaneously sorts in each of the 4 directions (right, right-down, down, left-down) to simplify the procedure, every update on the cell list is dynamically shared with sorting threads on the other directions (because the parsing entity is a pointer and it works directly on memory addresses). Thus upon completion, the end result is a neatly sorted list in every direction, and the total processing time is proportional to $4N$.

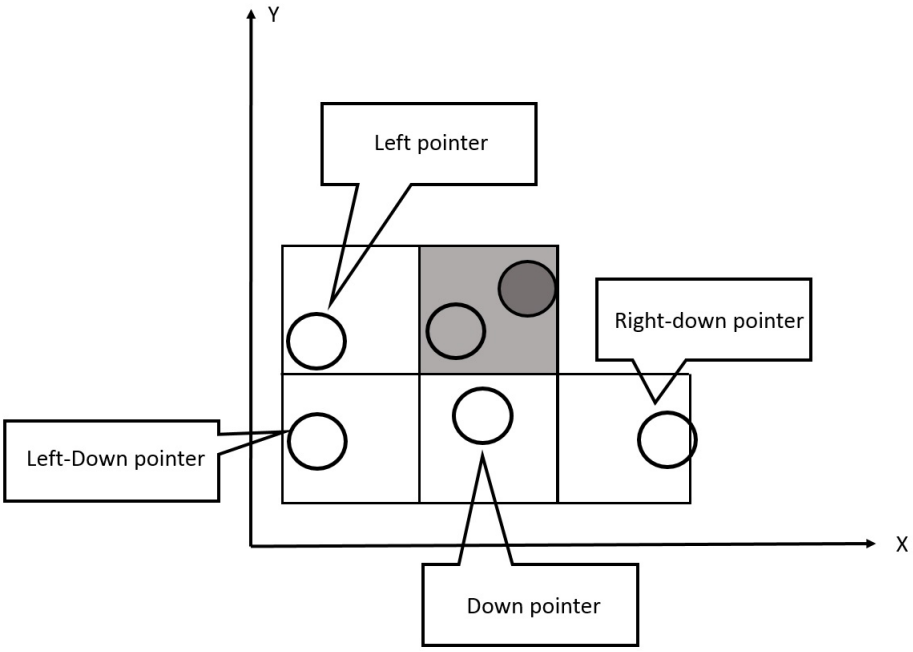

Figure 2.3: To sort the current contactor atom into its rightful position in the cell list, four pointers are used.

2.2.2.4 Contact Interaction

Under the context of rarefied argon gas, the interatomic interaction between the argons is a non-bonded one, which could be described by the Van der Waals function. The Van der Waals function is characterised by an attractive term and a repulsive term. One of the most popular Van der Waals force field is the Lennard-Jones $12-6$ function, given by

$$u_{vdw}(r) = 4\varepsilon[(\frac{\sigma}{r})^{12} - (\frac{\sigma}{r})^{6}] \tag{2.39}$$

in which, ε is the depth of the potential well, σ is the collision diameter.

In this study, the cut-off radius is chosen as $r_c = 2.5\sigma$. The interatomic energy decreases quickly as the distance increases. Introducing the concept of r_c is important to computational efficiency because it can greatly save computational power and boost overall computational efficiency. For particle pairs with an interatomic distance greater than the r_c, the pair is considered not in contact therefore subsequence collision computation could be spared. However, with the introduction of r_c, the potential model needs more modification to avoid the sudden jump of interaction force at exactly r_c.

One such model is called the Shifted Force Model, given by

$$\begin{aligned} u_s(r) &= u(r) - u(r_c) \quad , r \leq r_c \\ u_s(r) &= 0 \qquad\qquad , r > r_c \end{aligned} \tag{2.40}$$

in which, $u_s(r)$is the shifted potential, $u(r)$ is the potential in the original force field model. The function used in this study to represent the Van der Waals potential is the Buffered $14-7$, proposed by Halgren. The advantage Buffered $14-7$ has is that it prevents the potential going to infinity when the interatomic distance approaches zero and therefore has a more accurate repulsion term for the Van der Waals potential. The Buffered $14-7$ is given by:

$$u_{vdw}(r) = \varepsilon_{ij}\left(\frac{1+\delta}{\rho_{ij}+\delta}\right)^{n-m}\left(\frac{1+\gamma}{\rho_{ij}^m+\gamma} - 2\right) \tag{2.41}$$

$$\rho_{ij} = \frac{r_{ij}}{r_{ij}^*} \tag{2.42}$$

In which, ε_{ij} is the well depth, rij is the interatomic distance, the r_{ij}^* is the minimum energy distance. And according to Halgren, for rarefied gas, $n = 14$, $m = 7$, $\delta = 0.07$, $\gamma = 0.12$, therefore the Buffered $14-7$ in this study is given by

$$u_{vdw}(r) = \varepsilon_{ij}\left(\frac{1.07r_{ij}^*}{r_{ij}+r_{ij}^*}\right)^7\left(\frac{1.12r_{ij}^{*7}}{r_{ij}^7+0.12r_{ij}^{*7}} - 2\right) \tag{2.43}$$

From the potential function, the interatomic force can be derived at

$$f = -\frac{\partial u(r)}{\partial r} \tag{2.44}$$

This interatomic force is used to update the atom velocity in every time step.

$$v_t^{new} = v_t^{old} + \frac{f_t}{m} \times \triangle t \tag{2.45}$$

in which, v_t^{new} and v_t^{old} are the velocity after and before the update at time t, f_t is the instantaneous interatomic force acted on the atom, m is the atomic mass, and Δt is the time step.

2.2.2.5 Boundary Conditions

There are two perspectives to deal with the boundary contact problem (also called the boundary conditions). The boundary could either be treated as aggregation of particles, or as a wholesome boundary. For most casaes, the second approach can yield proper results and demands less computational effort, however the first approach should be employed whenever the atom-surface interaction could greatly affect the system kinetics. In this study, the second approach is used.

Having decided the boundary type, the second question is to define under what conditions atoms can be considered contacting with a certain boundary. A certain distance needs to be marked that within which, atom-boundary contact takes place and beyond which, atoms can be considered to be free bound by the boundary. Therefore, the potential function is derived by integrating all the atomic-boundary potential existed within a cut-off distance perpendicular to the boundary. Individual atom's influence on the boundary is measured by the contact distance d_{aw}, from the atom to the boundary.

In this thesis, a common potential field called Weeks-Chandler-Anderson potential (WCA potential) is employed. The backbone of WCA potential is the Lennard-Jones $12-6$ potential, however, WCA only retains the repulsive part, which is plausible enough in the boundary contact scenario. The WCA potential is given by

$$\begin{array}{lll} u_{WCA}(d_{aw}) = & 4\varepsilon[(\frac{\sigma}{d_{aw}})^{12}-(\frac{\sigma}{d_{aw}})^{6}]+\varepsilon & , d_{aw} < 2^{1/6}r \\ u_{WCA}(d_{aw}) = & 0 & , d_{aw} \geq 2^{1/6}r \end{array} \quad u_{WCA}(d_{aw}) = \tag{2.46}$$

in which ε is the depth of the potential well, σ is the collision diameter, and the cut off distance is chosen at $2^{1/6}\sigma$ in this study.

Therefore, the key job in calculating the boundary potential field lies in determining the d_{aw}. In cases with planar walls and regular boundaries, this can be easily solved, however in the context of irregular boundaries, special codes need to be implemented to speed up the process.

2.2.2.6 Integration of Motion Equations

In an MD simulation, each particle is treated as a node. Newton's second law of motion governs the interaction between the node and the rest of the nodes. With temporal discretization of the governing equation, the trajectory of each particle can obtained. To this end, at each time step, the particle coordinates (x_i) and the particle velocity (v_i) are collected and updated accordingly, this process is crucial for obtaining the particle trajectory. At each time step, inter-particle forces (f_i) and particle acceleration (a_i) and

velocity after impact can be calculated from particle position and initial velocity.

$$x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix}, \; v = \begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \\ \vdots \\ \dot{x}_n \end{bmatrix} \tag{2.47}$$

$$a = f_{ia}(F_{iexternal}, m_i) \tag{2.48}$$

in which, $F_{iexternal}$ is the sum of the external forces on a particular particle at a particular time, m_i is the particle mass.

There exists two kinds of time integration scheme in a typical MD system, namely the implicit time integration scheme and the explicit time integration scheme. One drawback of the implicit scheme is that it is inefficient in dealing with large systems, it is more comfortable with systems comprising of several hundred particles. In this study, the explicit time integration scheme is used. There are a range of different explicit schemes, for instance, Central Difference,[5] Position Verlet,[163] Forest&Ruth,[52,116,115] to name just a few. In this study, the Position Verlet scheme is used.

The Position Verlet (PV) Time Integration Scheme is given by

$$r_{t+\triangle t/2} = r_{t-\triangle t/2} + \triangle t \times v_t \tag{2.49}$$

$$v_{t+\triangle t} = v_t + \triangle t \times \frac{\mathbf{F}_{t+\triangle t}}{m} \tag{2.50}$$

in which, the $r_{t+\Delta t/2}$ is the particle position at time $t + \Delta t/2$, the $r_{t-\Delta t/2}$ is the particle position at time $t - \Delta t/2$, the v_t is the particle position at time t, the $v_{t+\Delta t}$ is the particle position at time , the $F_{t+\Delta t}$ is the external force acting on the particle at time $t + \Delta t$, and m is the particle mass. Noted that, the external force can be obtained from the inter-particle potential field function which takes the inter-particle distance as an argument.

2.2.2.7 Working Units

The working unit used in this study is given as follows:

$$time \rightarrow T = 10^{-12}s \tag{2.51}$$

$$length \rightarrow L = 10^{-10}m \tag{2.52}$$

$$mass \rightarrow M = 10^{-26} kg \tag{2.53}$$

The reason for using this set of scaled unit is simply for the sake of clarity and to avoid typos. An example is offered below to explain why un-scaled (natural) unit might be a nuisance. For argon atoms,

$$mass = 6.6335 \times 10^{-26} kg \tag{2.54}$$

$$\sigma = 3.405 \times 10^{-10} m \tag{2.55}$$

if, two argons atoms are separated by a distance of $4nm$, therefore the Lennard-Jones potential and force for the argon pair can be calculated as

$$u(r) = 4\varepsilon[(\frac{\sigma}{r})^{12} - (\frac{\sigma}{r})^{6}] \tag{2.56}$$

$$f = \frac{24\varepsilon}{\sigma}[2(\frac{\sigma}{r})^{13} - (\frac{\sigma}{r})^{7}] \tag{2.57}$$

in which,

$$\varepsilon = 161.0 \times k \tag{2.58}$$

$$k = 1.38065 \times 10^{23} J/K \tag{2.59}$$

Replacing eq.2.58 and eq.2.59.into eq.2.57.

$$f = \frac{24 \times 2.2228 \times 10^{-21}}{3.405 \times 10^{-10}}[2(\frac{3.405 \times 10^{-10}}{5 \times 10^{-10}})^{13} - (\frac{3.405 \times 10^{-10}}{5 \times 10^{-10}})^{7}] \tag{2.60}$$

As can be seen, it would make code-writing easier and computation easier if a simple scale is employed. Also the force, pressure and energy unit are given as follows:

$$force \rightarrow F = 10^{-12} N \tag{2.61}$$

$$pressure \rightarrow P = 10^{-8} Pa \tag{2.62}$$

$$energy \rightarrow E = 10^{-22} J \tag{2.63}$$

Therefore, the eq.2.57 can now be neatly re-written as

$$f = \frac{24 \times 222.28}{3.405}[2(\frac{3.405}{5})^{13} - (\frac{3.405}{5})^{7}] \tag{2.64}$$

2.3 Parallel Structures

The parallel computing structure could be classified by the Flynn's taxonomy:[62, 73, 76, 82, 1, 94]

- Single Instruction Stream-Single Data Stream (SISD). For example, a classic von Neumann system.[101, 104, 87]
- Single Instruction Stream-Multiple Data Stream (SIMD). SIMD systems are characterized by executing the same instruction on multiple data items. For example, the vector processors and GPU.[91, 90, 130, 133, 139, 164]
- Multiple Instruction Stream-Multiple Data Stream (MIMD). For example, the shared memory systems and distributed memory system.[112, 113, 114, 105, 104]

2.3.1 SISD Systems

The most important feature of an SISD system compared to parallel computing is that at any time, the CPU could only do one thing, it could either write to or read from the main memory, but not both. Thus the speed of the information exchange between the CPU and the main memory is the bottleneck of an SISD system.

Nowadays, most computers are manufactured with additional memory unit called cache besides the main memory. The CPU could read from and write to cache at the same time, the information exchange speed between CPU and the cache is much greater than that between the CPU and he main memory.[112, 49, 50, 59, 12]

2.3.2 SIMD Systems

Conventional CPU performs instructions on single piece of data, while vector processors could perform the same instructions on the whole vector of size n without any loop required. Workload is first shared among processors, each processors then perform the same instructions on their assigned workload. This type of parallelism is very useful for speeding up numerical simulations.

Vector machines have been successfully utilized for parallelization of Molecular Dynamics simulations.

2.3.3 MIMD Systems

Shared-Memory System

Parallelisation efforts in the field of methods of discontinua directed at the shared-memory systems have been significant due to the widespread use of multicore processors in recent years. These efforts include MD simulations[43,102,156] as well as DEM simulations.[109,9,170] Performance comparisons between the shared-memory system and the distributed memory systems have been studied extensively in both MD and FDEM. Hu and Lu[72] and Brown and Sharapov[16] have carried out performance comparison of a parallelized MD code in both shared-memory system (using OpenMP) and distributed-memory systems(using MPI). In FDEM, Owen and Feng[118] and Schiava D'Albano[38] carried-out a performance comparison of a parallel DEM code designed for parallelization on systems using hybrid of OpenMP and MPI on clusters have also been studied extensively.[30,31]

A shared-memory system consists of multiple processors and only one memory which is shared among all processors. Each processor also has its own cache which reads and writes data from into the main memory, the data in the cache is private from other processors. In a sense, all processors share the main memory, but the working data is first stored in its cache, which updates the main memory in an unpredictable fashion unless been explicitly asked to.[37,39,40,41,81,82,121]

Therefore when dividing work in a shared-memory system, it is important to identify independent variable from dependent variables. And for dependent variables, it is crucial to make sure that they are represented consistently across all processors.[23,79]

OpenMP (Open Multi-Processing) is an API that supports multi-platform shared-memory multiprocessing programming in C, C++ and FORTRAN. The core element of OpenMP is its workload distribution (work sharing), data environment management and thread synchronization. In C/C++, OpenMP uses directives like #pragmas to created parallelized sections incrementally.

Distributed Memory System

A distributed-memory system consists of a number of processors each has its own main memory. Processors communicate with each other by explicitly sending and receiving messages between each other, this is called message passing.[125,104,147,172]

A typical HPC cluster falls into the category of distributed-memory MIMD system. Each computer in the HPC is called a *node* which may contain more than one processor. Thus a typical HPC cluster is a *hybrid system* of a distributed-memory network consisted of shared-memory computers (nodes).

The main disadvantage of a distributed-memory system is the overhead cost of communication. In contrast, communication overhead cost in shared-memory system is much lower because memory is shared among all processors.

Message Passing Interface (MPI) is a standardized message passing system designed for parallel computers. MPI is a language independent communication protocol, which supports both point-to-point and collective communication.

MPI has been actively implemented in a range of MD simulation codes for parallelization.[7,57,164,84]

2.4 Parallelization Solutions to Molecular Dynamics Simulations

The two main objectives of parallelisation are to have a balanced workload (equal problem size) on each processors and low communicational overhead cost (that the amount of data exchanged between processors during communication should be as small as possible).

Recent years have seen the increasing application of MD simulations from material science to biophysics and to quantum physics.[42,54,71,97,108,137,96,148,35,46,85,86] And as the complexity of the simulation system increases, so does the number of the simulation particles, sometimes millions of particles need to be simulated.[138,135,134]There have been considerable efforts to study the parallelization solutions of MD simulations.[110,141,51,90,106,26,87,122,145,167,113]The most characteristic feature of any MD simulations is a large number of separate bodies moving and interacting with each other. The distribution of these bodies within the computational domain is changing in an unpredictable way during the simulation runtime, therefore the communication across processors at the end of each time step is important for correct particle migration. The rest of the chapter is an overview of the most common domain decomposition techniques and their main features. For systems with short range force, there are predominantly three parallelization strategies in MD simulations,[100,155] namely the Replicated Data method (RD), the Atom Decomposition method, the Force Decomposition method (FD),[8,69] and Spatial Decomposition method (DD).[12,18,20,21,22]

2.4.1 Decomposition Methods

The traditional way[66,124,126,128] to parallelized the force evaluation of inter-particle contact are categorized as atom,[?] force and spatial decomposition. Each method aims to divide the whole simulation domain in a particular way. AD distributes (arbitrarily) the particles in the simulation domain evenly among processors, SD is similar to AD

but it differs from AD in that the particles in each processor is spatially close to each other therefore the inter-processor communication is completely local. In contrast, FD distribute the force matrix evenly among processors. In other words, AD aims to target the uneven balance of particles in the domain, FD aims to target the imbalance of force interaction densities.

In evaluating the different decomposition methods, it is important to evaluate the following aspect of each method:

- the communicational cost per processor,
- the communication-to-computation ratio,
- and the load balancing problem.

Method	Communication cost	Computational cost	commu-to-compu ratio	Programs
RD	$O(NlogP)$	$O(N/P)$	$O(PlogP)$	CHARMM, AMBER
AD	$O(N)$	$O(N/P)$	$O(P)$	EGO
FD	$O(N/\sqrt{P})$	$O(N/P)$	$O(\sqrt{P})$	LAMMPS, CHARMM
SD	$O(N/P)$	$O(N/P)^2$	$O(N/P)^{1/2}$	SIGMA, ddgmq

Table 2.1: For a 2D simulation, the scalability of different decomposition strategies.

The communication across processors is very expensive for distributed memory system, therefore the communicational cost per processor determines how quickly the parallelized version of the code could run.

The communication-to-computation ratio is a very important concept in parallel computing as it shows to how much extent is the parallelized code scalable with increasing number of processors.

The load balancing is very important in parallel computing as well because at each time step, synchronization need to be carried out across all processors. And if certain processors have a huge work load it will take them significantly longer time to do the job (and hence, to reach the synchronization point), as a result other processors will have to sit idle at the synchronization point to wait for the slower processors to finish the job (increasing computation time), thus bringing down the efficiency.

2.4.1.1 Atom Decomposition Methods

The Atom Decomposition Methods is an improved version on the one of the earliest parallelization method, Replicated Data. In the RD scheme, the N number of particles in the whole simulation domain is evenly distributed into P number of processors, each processor will calculate the force on its local particles.

At each time step, each processor will examine all N particles in the entire domain to decide the contact forces for its local N/P particles and then update the net force for its N particles.

Assuming a cutoff radius, the local computational cost scales with N/P. As the updated net force on all of the N particles need to be shared and added up across all processors, it leads to a communication time cost proportional to $NlogP$. Thus, the communication-to-computation ratio is $PlogP$. A computational-to-communication ratio that is independent of the number of particles N means the algorithm is not strictly scalable, because should the number of particles N doubles, it is not possible to retain the same efficiency by doubling the number of particles P. The biggest advantage of RD is that it is relatively easy to implement, the biggest downside is that is it is not scalable.[124] RD is used widely in commercial MD algorithms like CHARMM,[19] AMBER,[168] UHGromos and etc.

Like RD, AD also assigns each processor with a fixed group of particles. In a system with N particles, each of the P processor is assigned a group of N/P particles, the particles assigned to each processor need not to have any spatially relationships, one common way is to group the same type of particles in one processor. The only difference lies in the communication mode.

In each time step, each processor will compute the force acting on its N/P particles by examining its local particle's distance to all the particles in the whole simulation domain, this means the computational cost of each processor is $O(N/P)$. This also means that at each time step, each processor will need a copy of the position of all other particles in the domain across all the other processors. The message size scales with N, instead of the point-to-point communication mode deployed in RD, AD employs an all-to-all communication mode which leads to an $O(N)$ communicational cost per processor, thus producing a computation-to-communication ratio at P. Therefore, AD is not strictly scalable and works most efficiently with a huge number P of processors. A number of commercial MD algorithms use AD to parallelize, e.g. EGO and etc.

In terms of load balancing, each processor will have an equal amount of work if the simulated system is has a uniform atom density. However, density non-uniformity

might arise in cases like phase change and etc. This could be overcome by randomly permutating the atom ordering in the beginning of each time step.

Various algorithms have been developed to perform the communication efficiently on different parallel machine and architectures, usually the optimized all-to-all communication algorithm provided in the MPI library is used.[83]

In summary, the atom decomposition divide the force calculation evenly across processors. The advantage of AD is that it is easy to implement, very few changes are required to parallelize an existing sequential code, this characteristic makes AD the most popular technique for parallel molecular dynamics.[128,127,147,158,162,100] The essence of the AD method is that each processor runs the same MD program, performing the same operations up to the point where a task (the force evaluation) need to be carried out in parallel, and each node takes part of the parallel task, and at the end of this task any data required by all processors must be passed and shared in a communication step. The draw back of the AD is that first, it is relative expensive in terms of memory as each processor needs a complete copy of all the particles in the simulation domain,[25,24,26,27,34,32] and second, this method also requires global communication across the processors, the communicational cost scales with N, and is independent of P the number of processors, leading to a high communicational cost, also the performance is not scalable. AD has been extensively used in a range of MD simulations.[18,49,70,77,95,133,143,150,149,152,151,171,10,66,162]

2.4.1.2 Force Decomposition Methods

FD methods a is block decomposition that distributes the work load of force calculation (the force matrix) across processors at each time step. The main difference between FD and AD is that FD decomposes the force matrix in blocks, while AD decomposes the force matrix in rows with respect to the contactor atoms.

In each time step, for a system with N particles across P processors, the whole force matrix is of size $N \times N$, each processor will have a share of the matrix of size $(N/\sqrt{P} \times N/\sqrt{P})$, meaning the computational cost is $O(N/P)$ for each node, and for each processor to calculate this sub-matrix, it will need the coordinates of $2N/\sqrt{P}$ particles from $\sqrt{P}$ processors, which means the communicational cost is $O(N/\sqrt{P})$, thus the communication-to-computation ratio scales with $\sqrt{P}$. As explained in the previous section, FD is not strictly scalable.

Also, the load balancing might be a problem with FD. The processors will have equal share of work only when the force matrix block is sufficiently and randomly

permutated, in other words, the system has a uniform density.

To summarize, FD decomposes the force matrix in block-wise fashion. The main advantage of FD is that it is easy to implement, and its communication cost is relatively much low compared to AD. The disadvantage of FD is that it doesn't scale strictly.

FD methods is used a range of commercial MD software, e.g. LAMMPS,[128] CHARMM,[74] and etc.

2.4.1.3 Spatial Decomposition

The spatial decomposition divides the physical simulation domain into smaller subdomains (boxes) assigned to each processors.[33] In each time step, each processor will only compute the force and update the velocity of the group of particles that are spatially close to each other in the subdomain. Atoms will be re-assigned to different processors if they leave the subdomain of the local processor. In force calculation, since each processor will only need to assess the neighbouring particles that are immediately outside of its subdomain, it will only need to communicate with a handful (8 in $2D$ simulations, and 26 in $3D$ simulations) of neighbouring boxes that surround it. Thus, the communication in DD is completely local compared with the global ones in AD and FD. The key in SD scheme is to minimize the communication cost as low as possible, the communication cost is directly related to the length (surface) of the box. In a $2D$ simulation, the most commonly used communication scheme in SD algorithm is east-west-north-south exchange.

The biggest advantage of SD is its significantly lowered communication cost. It is achieved by exploiting the locality of particles within a domain by performing local communications. For a uniformly distributed $2D$ system, the computational cost scales with the surface of the box $(N/P)^2$, and the communicational cost scales with the perimeter of the box N/P. Therefore, the communication-to-computation ratio is derived at $(N/P)^{1/2}$, making the algorithm highly scalable with increasing number of processors.

One of the biggest disadvantage of SD is that it is not easy to implement, and in most cases the sequential code has to be re-written completely. Another huge disadvantage SD has to face is the possibility of load imbalance. Since the whole simulation domain is divided up spatially, if the simulated system is not uniformly distributed, some processors will end up with more particles than others, and since force evaluation is the most time consuming part in a sequential code, even a tiny difference in the number of particles across processors will lead to huge idle time for the rest of

the processors, thus reducing the efficiency. One way to solve the problem is to incorporate the base idea of Atom Decomposition which is to distribute a fixed number of particle to each processor with the base idea of Spatial Decomposition which is to group spatially close particles in each processor for lower communicational cost: Load Balancing.

Load balancing is a dynamics process, the aim of which is to make sure that at any given time step, each processor will have approximately the same work load to accomplish thus wasting no computational power. Before the end of each time step, after particles have updated their positions, if there exists particle migration, then the entire domain needs to be re-evaluated and re-divided in a way that each box will have the approximately same number of local particles.[69, 111] Dynamic load balancing needs to be carried out at the end of each step should there be particle migrations.

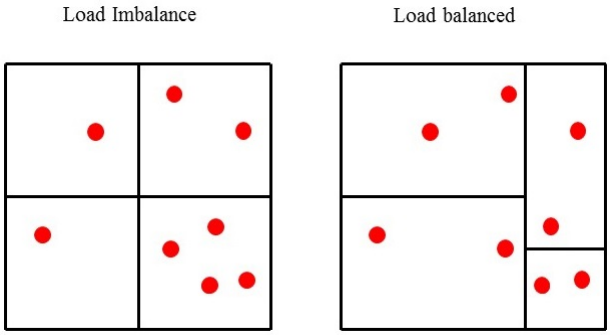

Figure 2.4: The dynamic load balance. Before load balancing, each processor handles a different number of particles ranging from 1 to 4, this will lead to huge waste of computational power as processors will have to halt and wait at the synchronization point. The dynamic load balancing solves the problem by redistributing approximately equal number of particles into each processors.

Recent years have seen a number of hybrid decomposition methods among the traditional AD, FD and SD.

Snir[153]and Shaw[144] independently developed the Neutral Territary Methods, albeit distinctly different, they are both a hybrid between the traditional Spatial Decomposition and Force Decomposition to parallelize the pair-wise range-limited (or distance-limited) particle interaction with a low communication band-wdith (the size of the message to be exchanged between processors).[16, 15, 80, 129] Not entirely like the SD method, the Neutral Territory Method sometimes evaluate a pair-wise interaction in a processor in which neither party of the pair resides. Several similar methods have been subsequently developed using the same techniques, for example, Bowers et al[16, 15]

have proposed the mid-point method, Bowers et al[14] have developed the eighth shell methods. The Neutral Territory Methods have been used extensively in bio-molecular simulations,[154] transient system analysis[129] and etc.

NAMD and NAMD2[107] have also been developed as unique hybrid decomposition strategies that combines the advantages of spatial decomposition and force decomposition, which permits the program to utilize a large number of processors.[123,98]

In this thesis, the decomposition strategy chosen is a Spatial Decomposition one. The reason being AD and FD which thrives on THE randomly permutation of either the particle list or the force matrix respectively will totally ruin the MR Contact Algorithm that exploits the near-sortedness and the spatial order of the particle list. Employing either AD or FD would render the MR sort and search useless with very little net gain in efficiency. The Spatial Decomposition method however is inherently coherent with the MR Contact algorithm, the idea is that the YNANO program should be parallelized in such a way that each processor could perform the MR sort and search algorithm independently in its local domain, the only thing alien to the original sequential program is the inter-processor communication.

2.4.2 Performance of a Parallel Implementation

The main aim to parallelise a sequential program is to enhance its performance. Thus, it is necessary to introduce some metrics to measure the performance of a parallel implementation, among many performance metrics, speedup and efficiency are the most commonly used.[82,89,158,171]

Speedup. Speedup is the ratio between the sequential execution time t_s and the parallel execution time (wall time) on p processors t_p

$$S = \frac{t_s}{t_p} \tag{2.65}$$

The speedup is considered linear when:

$$t_p = \frac{t_s}{p} \tag{2.66}$$

In practice speedup is usually smaller due to the expensive communication overhead cost between processors. By combining the above two equations, the speedup could be derived at:

$$S \leqq \frac{t_s}{t_s/p} = p \tag{2.67}$$

When $S > p$, the speedup is considered super linear. Super linear speedup is not impossible, but in most cases it is the result of utilizing extra amount of memory available on the parallel system. Therefore, in most cases the speedup is usually smaller than the number of processors p used. Communicational overhead is an unavoidable result from running a program on multiple processors. In the context of computing on a distributed-memory system, the overhead cost includes communication between nodes in the whole network. The speed of data exchange between nodes is usually much slower than the local memory access within a node. Thus shared-memory systems should have a much smaller communication overhead than the distributed-memory systems. Another overhead that comes with the Spatial Decomposition method used in Molecular Dynamics simulations is the particle migration. Since Molecular Dynamics simulations usually deals with a volatile system, and the nature of the Spatial Decomposition states that each processor only calculates a group of particles that are spatially close (ideally each processor has an equal number of particles), it is possible that at certain time step some particles will leave or entre the subdomain of a processor, and hence particle migration across processors is will need to be performed. The information that need to be exchanged in the particle migration process is the particle position, velocity and ID.

Efficiency. The efficiency of a parallel implementation is the ratio between speedup S and the number of processors p. The efficiency can be expressed as follows:

$$E = \frac{S}{p} = \frac{t_s}{p \cdot t_p} \tag{2.68}$$

Scalability. Scalability is a very important metric to assess the quality of a parallelized algorithm.[127, 104, 70, 59, 80, 13]

- Scalable: if efficiency E remains constant when both the number of particles N and the number of processors P increase in the same rate.
  - strong scaling: if the efficiency E remains constant when the number of processor P is increasing and the number of particles N is fixed. This means that the increasing communicational cost is well offset by the efficiency saved in computational cost.
  - weak scaling: if the efficiency E remains constant when number of processor P and the number of particles N increase at the same rate.

2.4.3 Floating Point and Rounding Error

Some fraction numbers need a large (or even infinite) amount of places to be expressed without rounding off. In a 32 bit system, a 32 bit variable can have 2^{32} different values, in a 64 bit system, a 64 bit variable has 2^{64} values. Therefore, any rational numbers (which is infinitely many) which is longer than 2^{64} do not have precise representation in the memory in a 64 bit system. In other words, due to the limited space of the memory and cache in a computer, it cannot accurately represent the infinite of these fraction numbers, therefore many numbers stored in the memory or cache are just approximations of the number they are intended to represent.

Real numbers have to be rounded off to be stored in the memory. However, this introduces a so-called rounding error which is an unavoidable consequence of the floating point computation.[61, 63, 44, 11] Any floating point number is saved in the computer's memory in the following format:

$$m \times b^e \tag{2.69}$$

where m is a significant (or mantissa), b is a base and e is an exponent. A significant is the part of a number in scientific notation or a floating-point number that consists of its significant digits p. A real number is rounded to the closest floating point number. The relative error of this operation can be calculated as follows:

$$relative\,error = \frac{real\,number - floating\,point\,representation}{real\,number} \tag{2.70}$$

When the real number is rounded to the floating point number, the maximum relative error of this operation is bounded by a so-called machine epsilon ε. The values of machine epsilon are prescribed by IEEE standard and they depend on the precision p, base b and the number of bits allocated in the memory.

Therefore, the rounding error has one very important unfavourable consequence for the parallel computation: Different processors will produce different results even by executing the same input file. This is because of the fact that, for instance, the summation of contact forces of a particle will be performed different orders on different processors, and with each summation comes with the rounding, in the end different order of summation will lead to different rounded summation for the same particle across processors.

With respect to this thesis, the rounding error could have a particularly nasty consequence in particle migration. Neighbouring processors may round the position of the

same shared particle differently, and this could lead to an disagreement to whether a particle should be migrated or not. For instance, at beginning of time t_0 two neighbouring processors P_A and P_B have a shared particle X, at the end of t_0, due to the rounding error on different machines, P_A decides that X has left its domain and it subsequently clears its particle information off its memory, but P_B decides that X is still in its domain. At time t_1, during force update between P_A and P_B, P_B will attempt to send to P_A the force components of X, but P_A no longer holds X in its domain, this could create a fatal error in MPI communication as the receiving message size doesn't match the sending message size, and will lead to the termination of the program.

Chapter 3

Development of a New Contact Detection Algorithm for Periodic Boundary Conditions

3.1 Introduction

The Molecular Dynamics simulation (MD) is an atomistic discontinua simulation tool which has been widely used in computational physics, biology, nano mechanic, nano technology and etc.[58, 88, 92, 131, 160] The most time consuming part of a MD simulation is its contact detection process, therefore a good contact detection algorithm is essential to the efficiency of a simulation.[1, 6, 18, 25] The MR linear contact detection (linear sort and linear search) algorithm is a widely-used contact detection algorithm because of its linear processing time proportional to the number of atoms. However, the MR is developed without a periodic boundary condition (PBC), which greatly limits its application in fluid and gas dynamics simulations.

The MR contact detection algorithm is a linear algorithm because it is built on the idea of sorting on a nearly-sorted list, which means that between two consecutive time steps, the movement of any particle is restricted under one cell length, this is also called the stability criterion. For example in Figure 3.1, during one time step, a particle in $[1,1]$ could only move within the 3×3 matrix immediately around it.

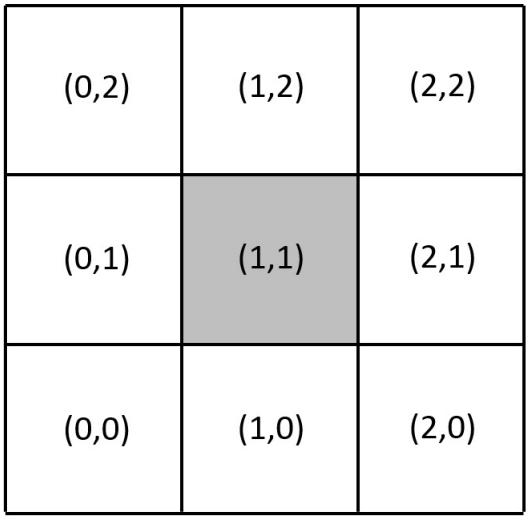

Figure 3.1: The neighbouring cells.

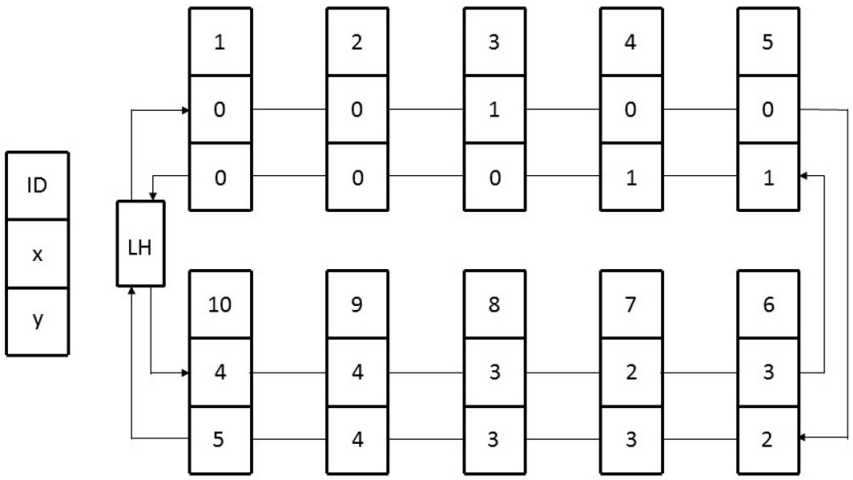

Figure 3.2: A perfectly sorted linked list.

The difference between a perfectly sorted and a nearly sorted list can be illustrated in Figure 3.2 and Figure 3.3. At time t_1, the list is perfectly sorted. However at time t_2, while all other particles maintain the same integerized coordinates, particle No.6 is no longer smaller than particle No.7 or particle No.8, therefore the list is no longer perfectly sorted, but rather, nearly sorted. The sorting process could be greatly enhanced if spatial relationship is preserved in a certain way between consecutive time steps. As discussed above, the order of list is only broken when a previously bigger (in

integerized coordinates) particle becomes smaller than its previous particle, there are four directions along which a bounding box can move to make this happen, namely, down, left, down-left, down-right.

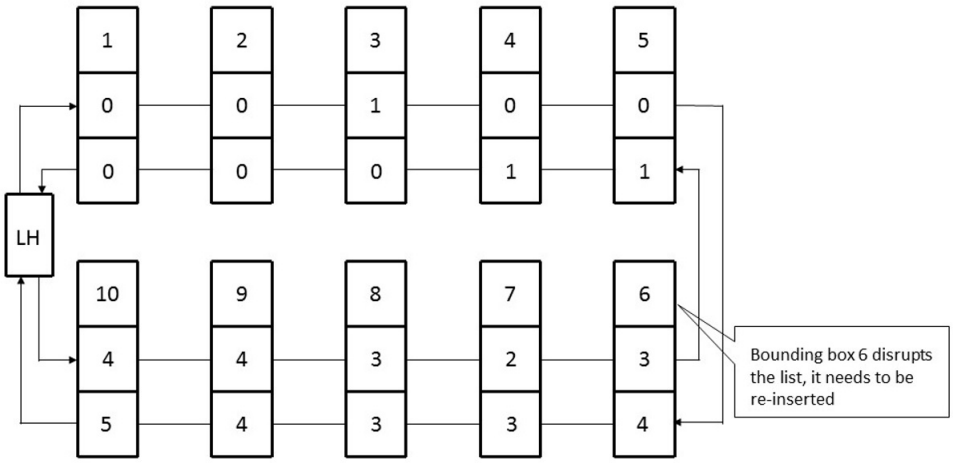

Figure 3.3: The linked list.

The MR_sort algorithm assigns four different pointers to track those directions, and as the list is parsed from the beginning to the end, each pointer is called forth when its corresponding moving direction is detected. The mechanism is very similar to the resolution of vector. As all pointers will only march forward once, in the end the list will only need to be parsed once to be updated with the right order.

As is evident from the above analysis, the MR_sort is a linear sorting algorithm, it fully ultilizes the stability criterion installed and exploits the spatial relationships of a nearly sorted list.

3.2 Problem Defined

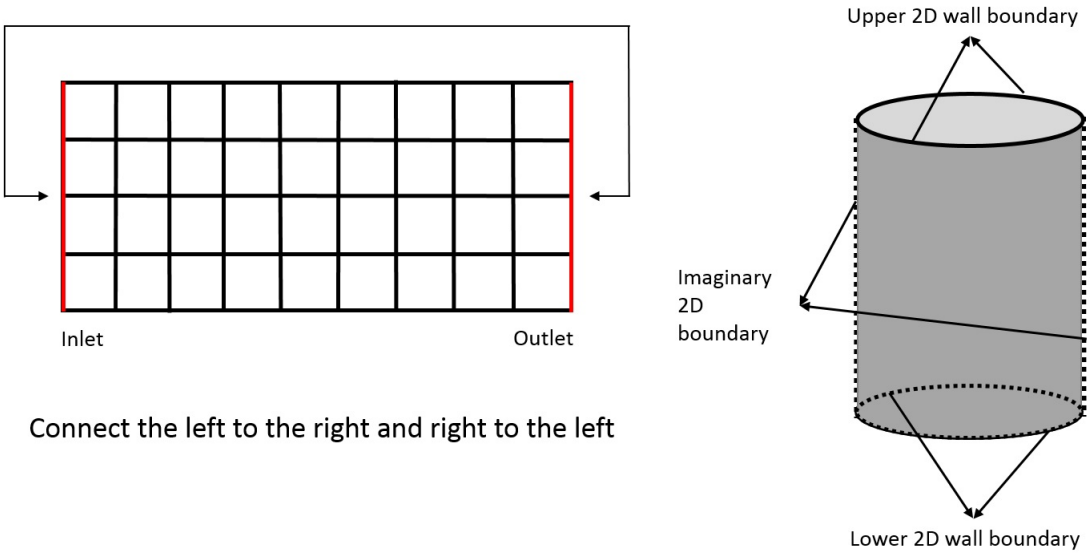

Figure 3.4: The periodic boundary condition scheme.

Periodic boundary conditions (PBC) can be used to simulate very big or infinite system. However, when it comes to applying PBC to the MR_sort and search algorithm, problems arise.

Firstly, the PBC breaks the stability criterion required by the MR_sort. Under PBC, when atoms on either end step out of the boundary, they will need to migrate back to other side of the domain, in most cases this step is greater than one bounding box length. As a result, the regular pointers in the MR_sort algorithm will keep going back and forth which might lead to one pointer repeatedly covering the part of cell list that it has covered before. As a consequence, the MR_sort for PBC might result in a sorting time far greater than linear time.

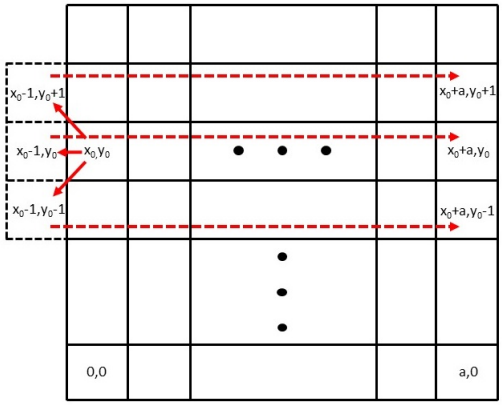

Figure 3.5: Problem arises under the periodic boundary conditions. When the dimension of the simulation domain is greater than one bounding box length, the stability criteria is broken.

In the physics sense, there is no problem moving the particles from one edge to the other as long as the energy of the system is conserved, the problem presented here is a computational sorting one resulted from coordinates indexing. The solution is, for all left periodic particles expecting to move to the right side, their integerized coordinates will be labelled in such a way that they are shifted one bounding box length up (one row up), and all right periodic particles expecting to move to the left side, their integerized coordinates will be labelled in such a way that they are shifted one bounding box length down i.e. one row lower.

For this labelling to work, the stability criteria needs to expand slightly into: between two consecutive time steps, no particle could move a greater length than twice the length of a bounding box. As a result, one extra directional pointer is needed to parse the list.

Given the fact that no ordering is kept for particles with the same integerized coordinates, the labelling is effectively fully utilizing the MR_sort to put the moved periodic particle right behind the particles in the MR list.

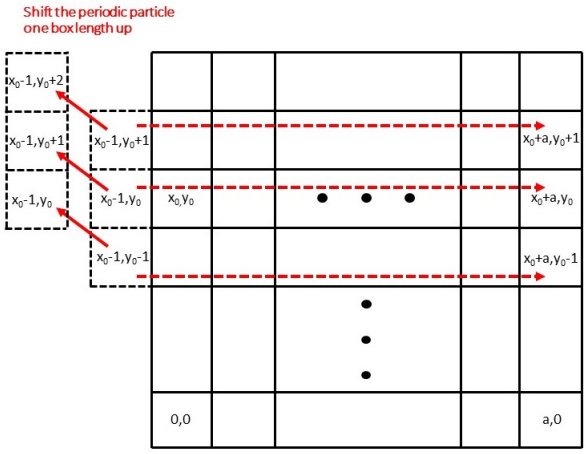

Figure 3.6: For all left periodic particles expecting to move to the right side, their integerized coordinates will be labelled in such a way that they are shifted one bounding box length up (one row up)

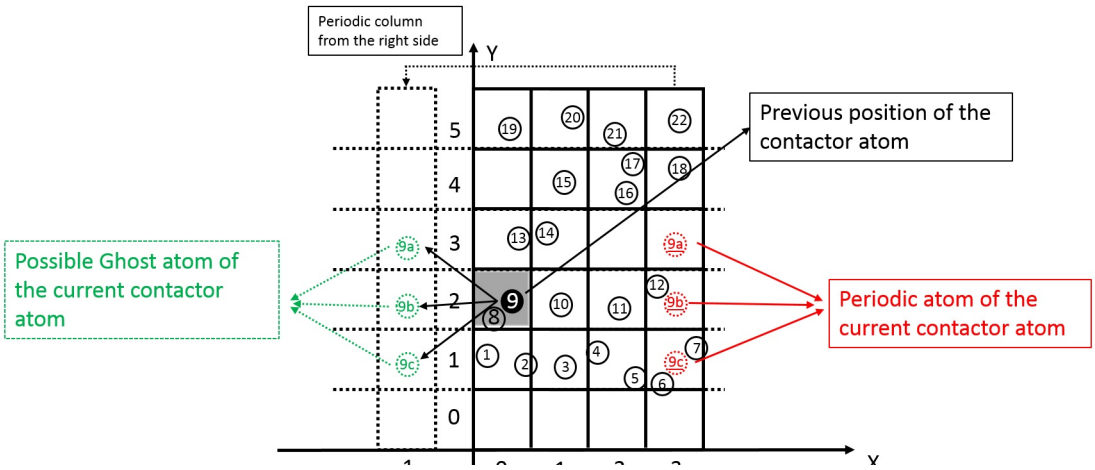

Figure 3.7: For atoms that step outside the left side boundary, No.9a, 9b or 9c are the possible ghost atoms of the current contactor atom; No.9a^*, 9$b*$ and 9$c*$ are the periodic atoms of the corresponding ghost positions.

As discussed above, the biggest obstacle in sorting a PBC cell list is dealing with the discontinuity of integerized coordinates for atoms that step in and out of the periodic boundary. In Figure 3.7, atom No.9 was in box $[0,3]$ in the previous time, in the current time it might move to be either atom No.9a, 9b or 9c, these are called the ghost atoms. Under the PBC assumption, atom No.9a, 9b or 9c should migrate to be the periodic atom No.9a, 9b or 9c on the other side. The discontinuity of integerized list lies in that the periodic atom No.9a, 9b or 9c now has larger coordinates than atom No.9a, 9b or 9c respectively, while in a physics sense, it should have acquired smaller

coordinates. This directly results in the inefficiency of the MR_sort algorithm, as in MR_sort algorithm, the advance pointers will have to move back and forth repeatedly to accommodate the PBC, as a result, the total running time will not be linear to the length of the list.

Thus, the problem can be termed this way: how to modify the MR_sort and search algorithm to achieve a (O) of linear time in sorting a cell list under the periodic boundary conditions. The key to solve this problem is how to make the periodic atoms that disturb the ordering of the list stay within order. In order to preserve the linearity of the sorting and searching algorithm, a new MR_PB linear algorithm is developed with a CPU time proportional to the number of particles.

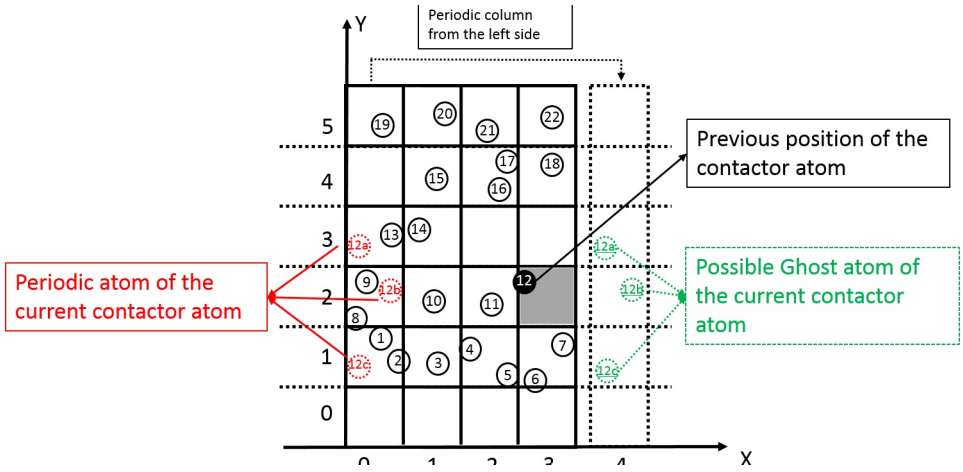

Figure 3.8: For atoms that step outside the left side boundary, No.12a, 12b or 12c are the possible ghost atoms of the current contactor atom; No.12a^*, 12$b*$ and 12$c*$ are the periodic screen atoms of the corresponding ghost positions.

3.3 The 2D MR_PB Linear Sort Algorithm

To solve this discontinuity, ghost-plugin atoms are created to help preserve the "nearly sortedness" of the cell list. Each ghost atom No.9a, 9b or 9c is shifted upwards by one cell size length to position 9$a*$, 9$b*$ and 9$c*$. This is equivalent to appending the ghost-plugin atoms right behind (larger than) the periodic atoms No.9a, 9b and 9c, and since ordering doesn't matter within a spatial cell, this shifting method can retain the "nearly sortedness" of the cell list.

It should be noted that in the MR_PB sort algorithm, only ghost-plugin atom No.9$a*$ will be parsed because it retains the spatial near-sortedness. Figure 3.9 illustrates a periodic atom on the other side (No.12). Atom No.12 was in box $[3, 2]$ in

the previous time, in the current time it might move to be Ghost position No.12$a*$, 12$b*$ or 12$c*$, which means the corresponding periodic atom would be No. 12a, 12b or 12c.

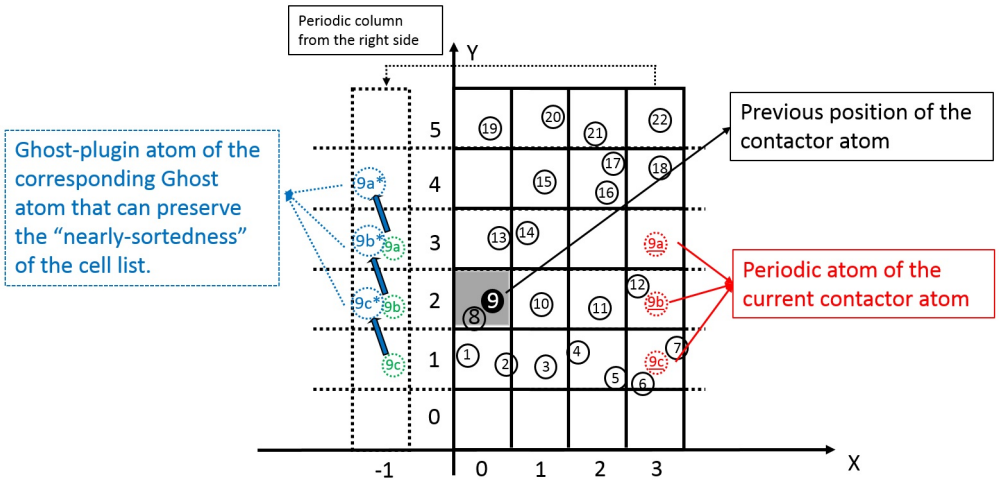

Figure 3.9: Each ghost atom is assigned with a ghost-plugin atom that can retain the spatial relation and preserve the "near sortedness" of the cell-list. For ghost atoms (No.9a, 9b or 9c) that step outside the left side boundary, ghost-plugin atom No.9$a*$, 9$b*$ or 9$c*$ each represents a bounding box that is one box-length higher than the ghost atoms bounding box. The end result is to place the pointer right after the periodic screen atom bounding box.

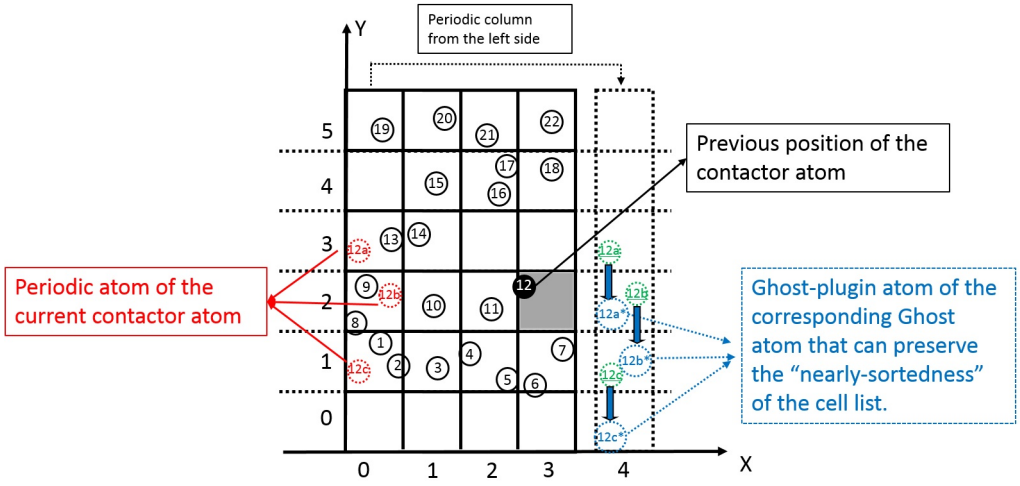

Figure 3.10: For ghost atoms (12a, 12b and 12c) that step outside the right side boundary, pointers 12a*, 12b* and 12c* each points to a bounding box that is one box-length lower than the ghost atom bounding box. The end result is to place the pointer right before the periodic atom bounding box.

Ghost-plugin atoms No. 12$a*$, 12$b*$ and 12$c*$ are created by shifting the ghost

atom downwards by one cell size length. This is equivalent to appending the ghost-plugin atoms right in front of (smaller than) the periodic atoms No.12a, 12b and 12c. Similarly in the MR_PB sort algorithm, only ghost-plugin atom No.12$b*$ and 12$c*$ will be parsed because they have retained the near-sortedness of the cell list. It should be noted that the MR_PB sort algorithm will be parsing on 3 additional pointers on top of the 4 pointers from the original MR_sort algorithm. Since every pointer will only parse forward once, the total CPU time will be proportional to the number of atoms.

Algorithm 3.1 Updating the ghost plugin particles.

```
1:  double dx                                   ▷ The x coordinate of contactor particle
2:  integer dgpx, dgpy                          ▷ Coordinate of the ghost-plugin particle
3:  double bnd_left, bnd_right                  ▷ X coordinate for left and right boundary respectively
4:  double size                                 ▷ The longitudinal size of the domain.
5:  double sizecell                             ▷ The cell size of the bounding box
6:  if (dx > bnd_right) then
7:      dgpx = dx + cellsize
8:      dgpy = dy − cellsize
9:      dx = dx − size
10: else if (dx < bnd_left) then
11:     dgpx = dx − cellsize
12:     dgpy = dy + cellsize
13:     dx = dx + size
14: end if
```

3.4 The 2D MR_PB Linear Search Algorithm

Contact is defined between two particles, the contactor and the target. For each contactor particle, its nearest neighbours are parsed to for potential contact pairs. A contact mask is created to avoid repetition in this process.

For contactor atom with minimum integerized x coordinate, that is, if it is on the left edge of the periodic boundary, its contact mask can be seen in Figure 3.11. The central cell will include not only the regular neighbouring cells, but also periodic neighbouring cells from the other end (with maximum integerized x coordinate). The contact mask consists of two rows, for each row, a beginning pointer (beg) and end pointer (end) are defined.

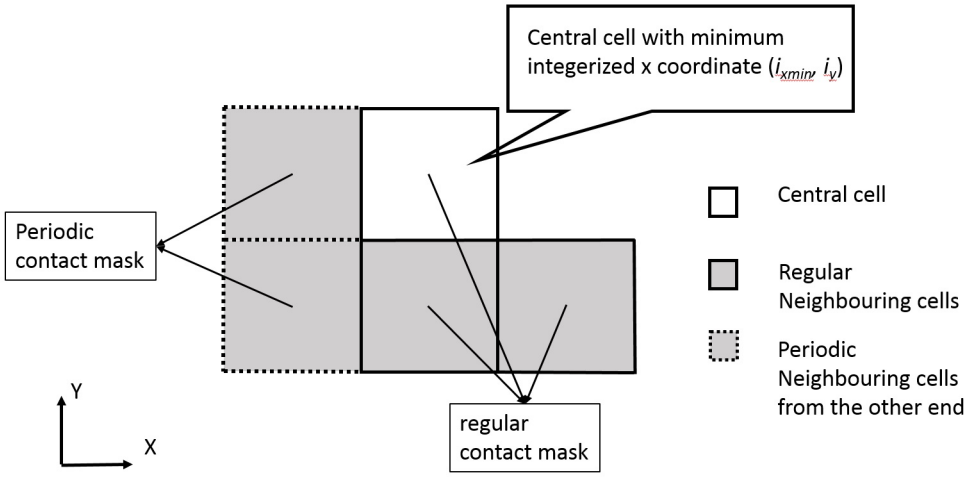

Figure 3.11: Contact mask in 2*D* for contactor atom with minimum integerized x coordinate.

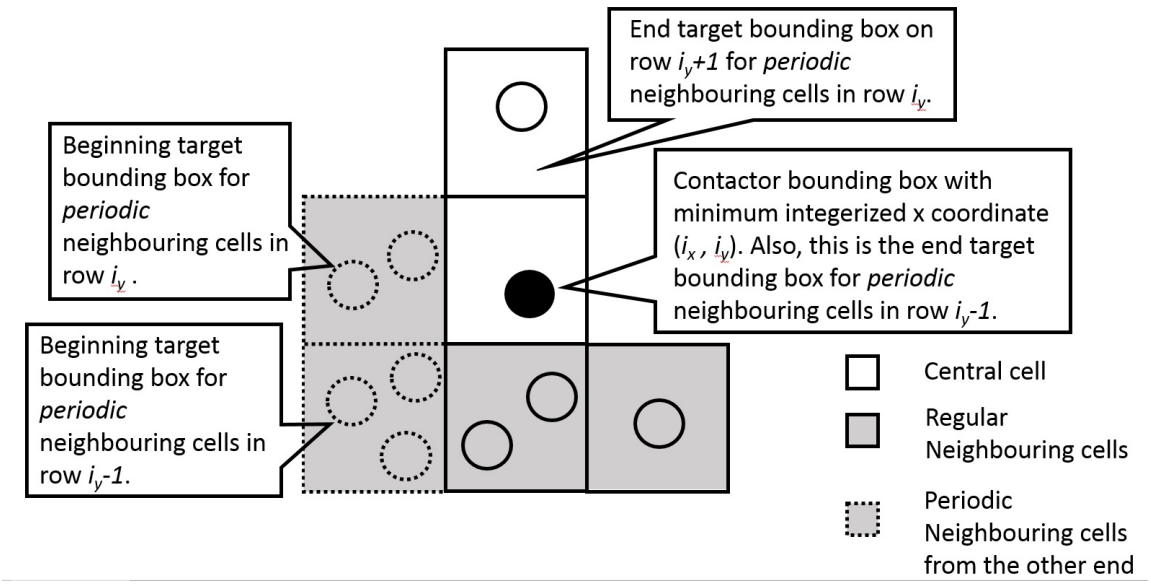

Figure 3.12: Contact mask in 2*D* for contactor atom with minimum integerized *x* coordinate and its target particle from the other side.

The aim of the process is to locate which atoms are in the neighbouring bounding boxes, the process is carried out by identifying which bounding boxes precede or append after each contact mask in each row. By parsing the cell-list from List Head (LH)

until a bounding box that is no less than the first box in each contact mask row, the beginning bounding box can be obtain. And by parsing from the beginning bounding box until a box that is greater than the last box in each contact mask row, the end bounding box can be obtained.

The regular contact mask is exactly the same as in the MR_search. For periodic bounding conditions, an additional periodic contact mask with two rows is implemented. In the case of contactor atom with minimum integerized x coordinate, two more sets of beginning and end bounding boxes are needed on top of the two regular sets, and in the case of contactor atom with maximum integerized x coordinate, one more set of beginning and end bounding boxes are needed for on top of the two regular sets.

By the same principle, for contactor atom with maximum integerized x coordinate, that is, if it is on the right edge of the periodic boundary, its contact mask can be seen in Figure 3.13.

In regular boundary conditions, two sets of beginning and end bounding boxes are needed to filter the cell list. However, in periodic bounding conditions, in the case of contactor atom with minimum integerized x coordinate, two more sets of beginning and end bounding boxes are needed on top of the two regular sets, and in the case of contactor atom with maximum integerized x coordinate, one more set of beginning and end bounding boxes are needed on top of the two regular sets.

A detailed example of the contact search and detection algorithm can be seen in Figure 3.15. The whole simulation domain is consisted of 16 bounding boxes, given the contactor atom as atom No.9 on the left edge and atom No.12 on the right edge, the contact search and detection for each contactor atom is explained as follows.

Pointer set 1 and 2 deal with the regular domain, pointer set 3 and 4 are for periodic particles on the left edge, and point set 5 is for periodic particles on the right.

First, atom No.9 is the current contactor atom. The regular neighbouring bounding boxes for atom are box $[0,2]$, $[0,1]$ and $[1,1]$, therefore the $beg1$ pointer will point to the first atom in box $[0,2]$ and $end1$ will point to the contactor atom itself. $beg2$ pointer will point to the first atom in box $[0,1]$ and $end2$ pointer will point to the first atom in box $[2,1]$.

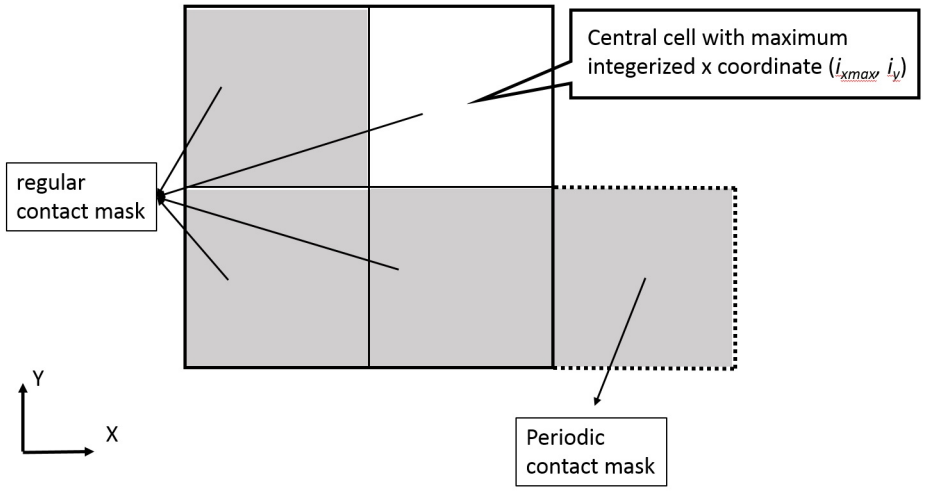

Figure 3.13: Contact mask in 2D for contactor atom with maximum integerized x coordinate.

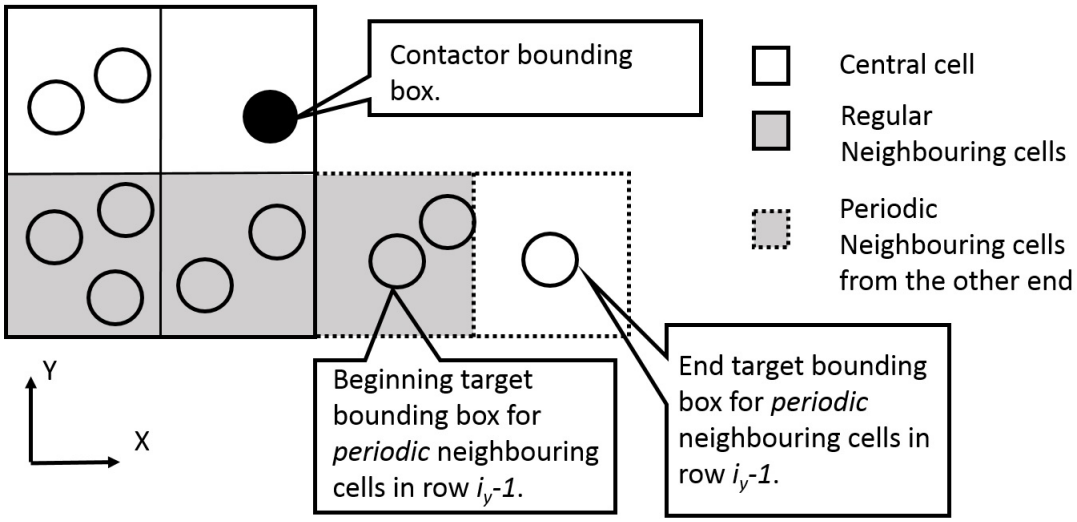

Figure 3.14: Contact detection in 2*D* for contactor atom with maximum integerized x coordinate and its target particle from the other side.

For periodic neighbouring bounding boxes, *beg*3 pointer will point to the first atom (No.12) in the last bounding box $[3,2]$ which is of the same line to the contactor atom, and *end*3 will point to first atom (No.13) in bounding box $[0,3]$ which is immediately after bounding box $[3,2]$. And finally, *beg*4 will point to the first atom (No.6) in the

last bounding box $[3,1]$ which is one line lower than the contactor atom, and *end*3 will point to first atom (No.8) in the contactor bounding box which is immediately after bounding box $[3,1]$. It is evident that for contactor atoms with minimum integerized x coordinate, 2 sets of extra pointer are needed.

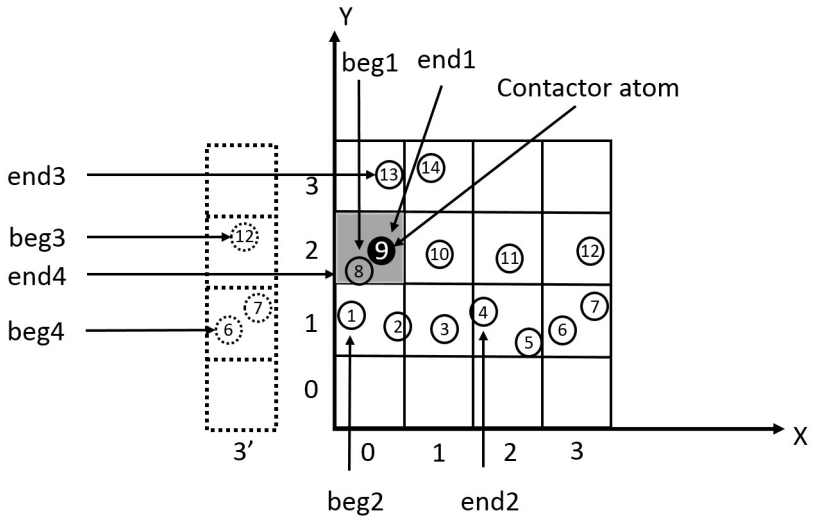

Figure 3.15: Contact detection range for contactor atom No.9 on the left side. Aside from the 2 sets of beginning and end pointers in the MR_search algorithm, another 2 sets of pointers need to be implemented to search for the cut-paste atoms on the other end.

For contactor atom on the right edge, for example atom No.12, as can be seen in Figure 3.16. It is evident from the scheme that atom No.1 and 2 are the periodic neighbouring atoms for the contactor atom. Therefore, for contactor atom with maximum integerized x coordinate, only one set of extra pointer is needed.

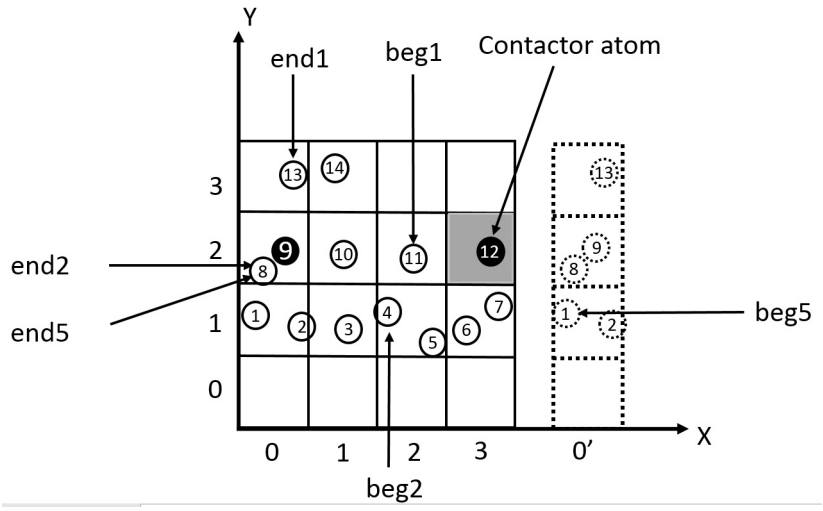

Figure 3.16: Contact detection range for contactor atom No.12 on the right side. Aside from the 2 sets of beginning and end pointers in the MR_search algorithm, another set of pointers need to be implemented to search for the cut-paste atoms on the other end.

After the completion of this stage, target atoms can be identified by selecting those atoms that are no less than the beginning bounding box and smaller than the end bounding box in each contact mask in each row. This way, contact search is simplified into parsing the atoms between each set of beginning and end bounding boxes. Since all boxes will only advance forward once along the ordered cell list, the total CPU time for MR_PB linear search is theoretically proportional to the number of atoms.

3.5 The 3D MR_PB Linear Sort Algorithm

In 3D simulations, the MR_sort is also built upon the stability criterion. However, the 3×3 2D matrix now needs to be expanded into a $3 \times 3 \times 3$ 3D matrix.

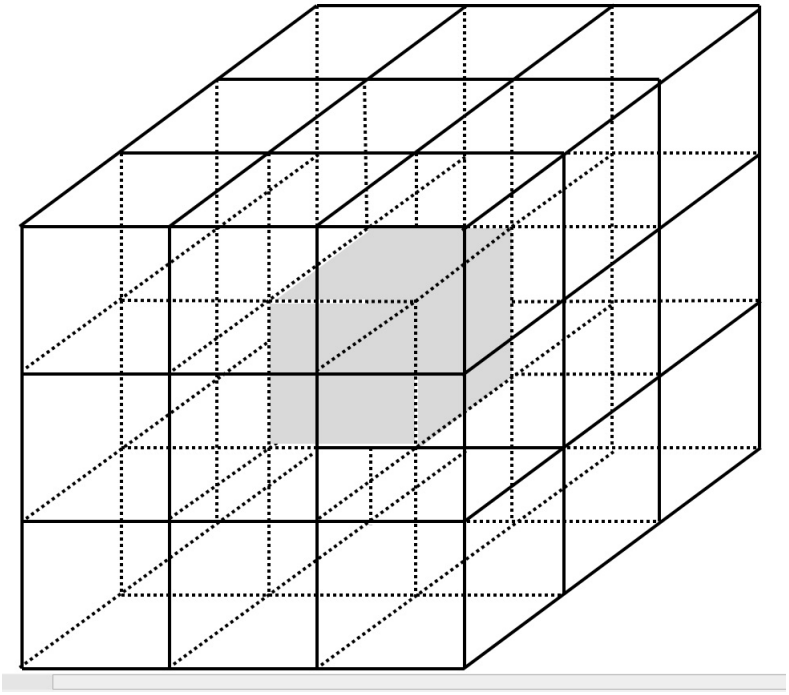

Figure 3.17: For a 3D simulation, the central cell now has 26 nearest neighbours.

For a 3×3 matrix of neighbouring cells in 2D simulations, it has been explained that four direction pointers will be used to track four directions of displacement. Similarly, in a $3 \times 3 \times 3$ 3D matrix, more direction pointers will be added to accommodate more directions of displacement, for the purpose of simplicity, all direction pointers are labelled alphabetically. In total, 13 direction pointers are needed in 3D MR_sorting.

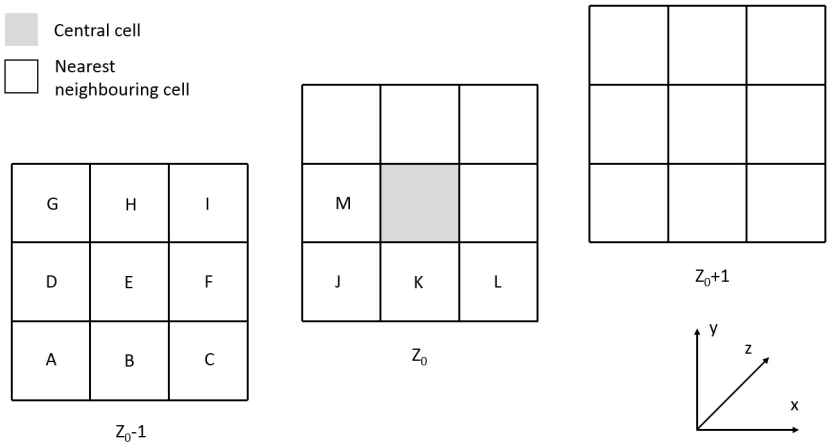

Figure 3.18: For the lightly shaded central cell, it has 26 nearest neighbouring cells around it, all are the possible location for its next step. Among the 26, only 13 (A to M) are of interest because these cells have smaller integerized coordinates than the central cell.

It should be noted that in the integerized coordination scheme deployed in the MR_sort and search, the *z* direction takes precedence over the *y* direction, and the *y* direction in turn takes precedence over the *x* direction.

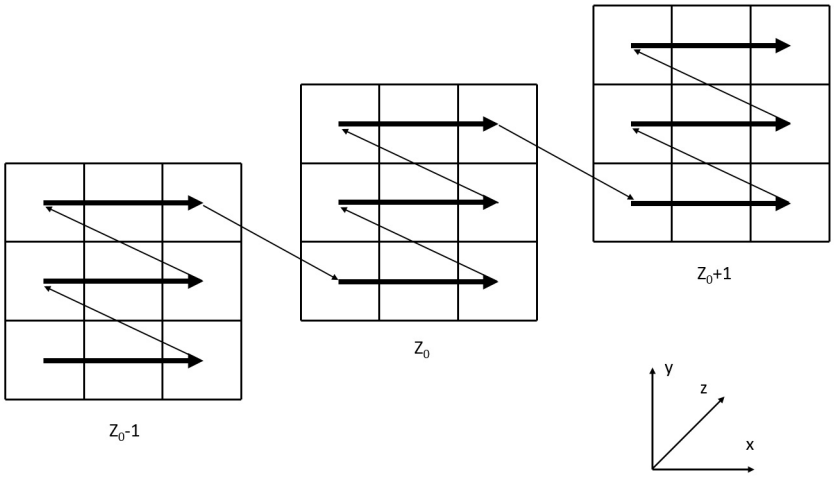

Figure 3.19: Under the integerized coordinate scheme, the *z* direction takes precedence over the *y* direction, and the *y* directions takes precedence over the *x* direction.

The precedence rule is important because it affects the choice of the longitudinal axis and periodic plane as inlet and outlet. For instance, if direction *z* is chosen as the longitudinal axis, and the *xy* plane is the periodic plane for inlet and outlet, then problem will arise.

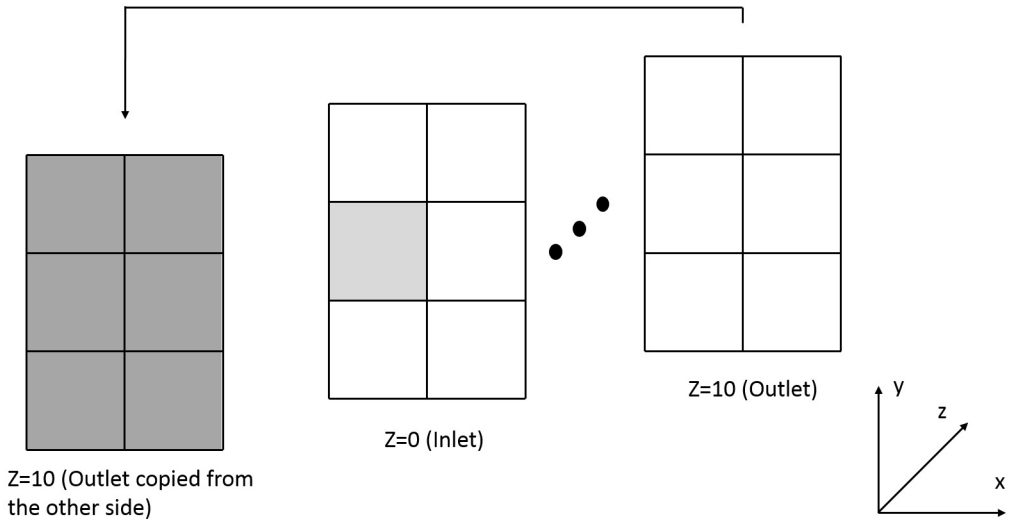

Figure 3.20: The inlet and outlet under periodic boundary condition.

The problem that arises could be summarized as: the periodic movement from the

edges (inlet and outlet) drastically promotes or demotes the periodic bounding box to the highest or lowest z hierarchy. Unlike the 2D case where the promotion or demotion can be mitigated by shifting up and down the y axis, because the drastic change takes place on the x axis, if the drastic change takes place along the z axis, it doesn't work here because already the z axis takes the highest precedence. Therefore for this study, the yz plane is chosen as the periodic plane, and the x axis is chosen as the longitudinal axis. In other word, the 3D PBC scheme will be an expansion of the 2D PBC scheme.

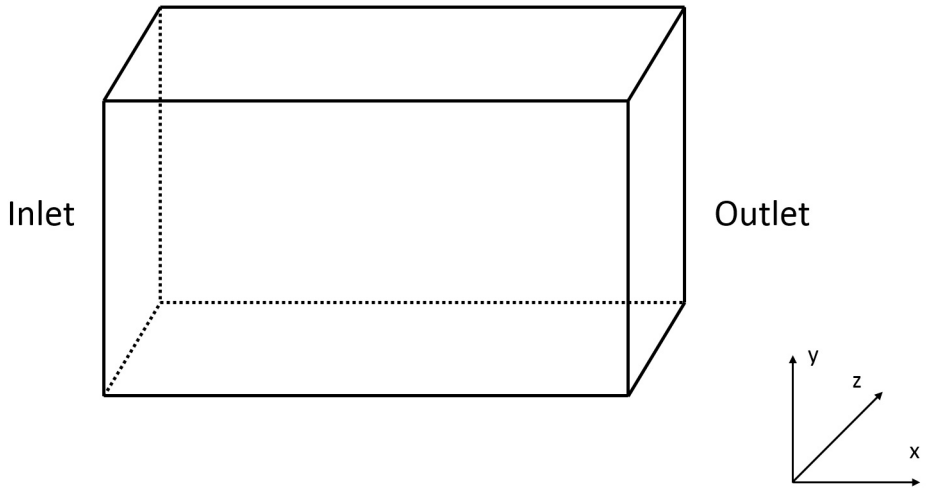

Figure 3.21: In this study, the yz plane will be the periodic plane and axis x will be the longitudinal axis. Thus, the 3D PBC will be an expansion of the 2D PBC.

As explained above and in the previous sections, the 2D MR_PB works around the problem by creating a set of corresponding ghost-plugin particles, they shift along the y axis either one cell size upwards or one cell length downwards, depending on whether the current particle is on the inlet or the outlet. The same solution can be applied to 3D PBC as well, since 3D keeps the basic structure of the 2D model.

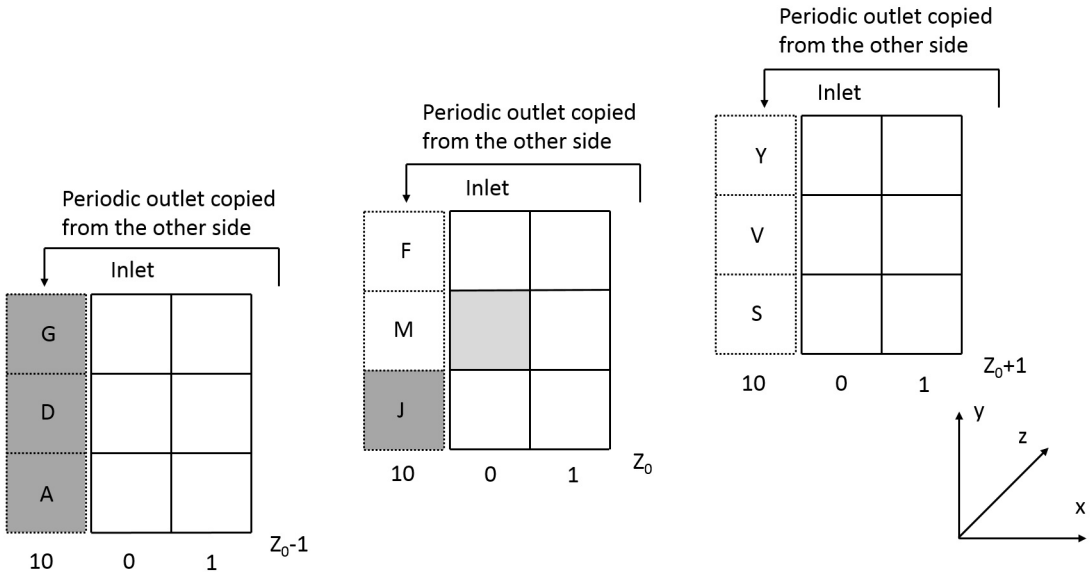

Figure 3.22: For current particle on the inlet box (lightly shaded) with minimum integerized x coordinate, it can move to any of the dashed boxes on the other side (copied from inlet). But only the heavily shaded ones will be re-arranged in the sorting process.

For instance, to create ghost-plugin particles for current particles on the inlet (with minimum integerized x coordinate), all the possible ghost particles are shifted upwards along y axis one cell size length, this is equivalent to appending the ghost-plugin particle right behind its corresponding periodic particle on the other side (on the outlet).

Similarly, for atoms on the outlet (with maximum integerized x coordinate), all possible ghost particles are shifted along the y axis one cell length downward, this is equivalent to inserting the ghost-plugin particle right in front of its corresponding periodic particle on the other side (on the inlet). Because ordering within one cell doesn't matter, this shifting method retains the spatial logic of the near-sorted list. It is evident from the analysis that two more pointers are needed.

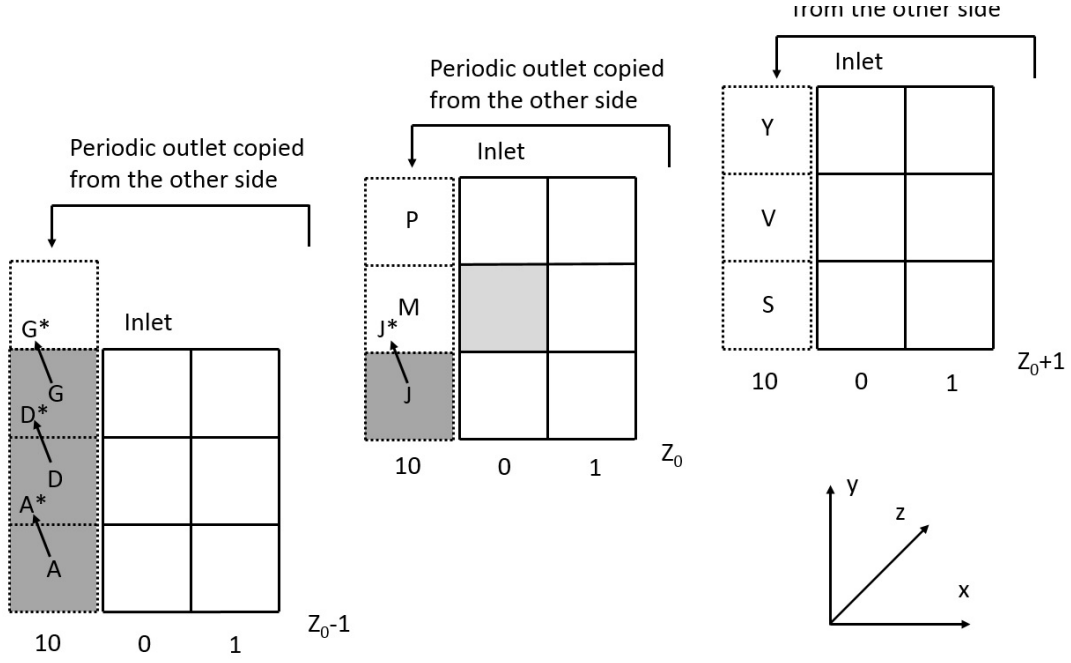

Figure 3.23: For current particles on the inlet, ghost-plugin particles are created by shifting the periodic particles one cell length upwards.

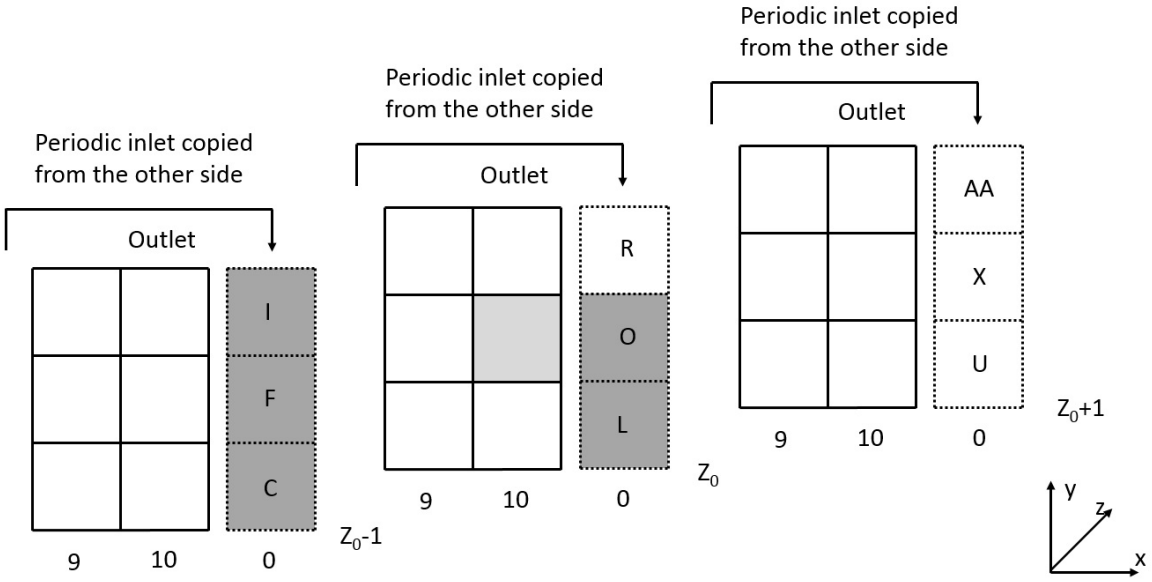

Figure 3.24: For current particle on the outlet box (lightly shaded) with maximum integerized x coordinate, it can move to any of the dashed boxes on the other side (copied from inlet). But only the heavily shaded ones will be re-arranged in the sorting process.

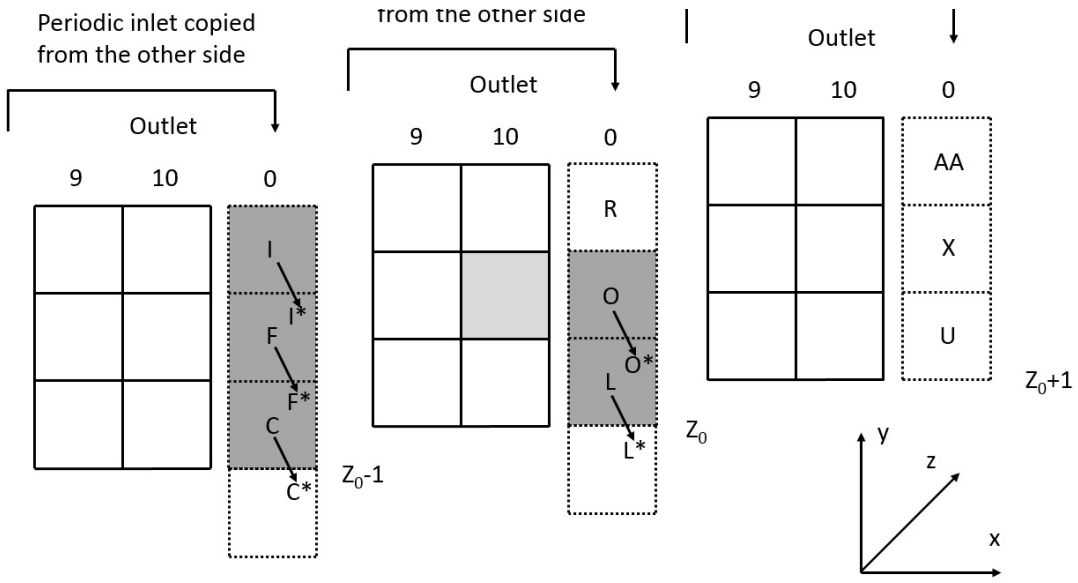

Figure 3.25: For current particles on the outlet, ghost-plugin particles are created by shifting the periodic particles one cell length downwards.

3.6 Verification of the MR_PB Linear Contact Detection Algorithm: 19,801 atoms inside a periodic cubic container

In this section, the MR_PB linear sort and the MR_PB linear search algorithm developed in the previous section are tested. The testing case is a $2D$ model. The system comprises of $19,801$ argon atoms placed in a periodic cubic container with rigid upper/lower walls and periodic left/right walls. The argon atoms are placed in a bcc (body centred cubic) configuration in the centre of the container in the start of the simulation, and each atom's velocity is assigned acoording to the Kinetic Theory. The upper/lower walls in the simulation are surface plane, the interaction between the atoms and the container walls is described by the Lennard-Jones $12-6$ interaction potential.

The total time steps taken are $100,000$ *ps* and the time step size of the simulation is $\triangle t = 0.001$ ps. After each contact between the atom and the wall, there should be no increment on the total internal energy, which is the sum of kinetic energy and potential energy. The total energy fluctuates during the collision between an atom and the wall, it is caused by the fact that during Central Difference time integration, atom velocity and the position are calculated at different point of time. For instance, the atom position is calculated at $t = t_i$, while its velocity is calculated at $t = t_i - \triangle t$. The

energy difference can be reduced by using a smaller time step.

The important fact is that total energy as the system is conserved as it relaxes into equilibrium, this shows that MR_PB sort and MR_PB search are accurate in finding all the contacting pairs at each time step. Because if the algorithm fails to identify a collision pair, there will be a jump in the system total energy, which is not observed in the test example.

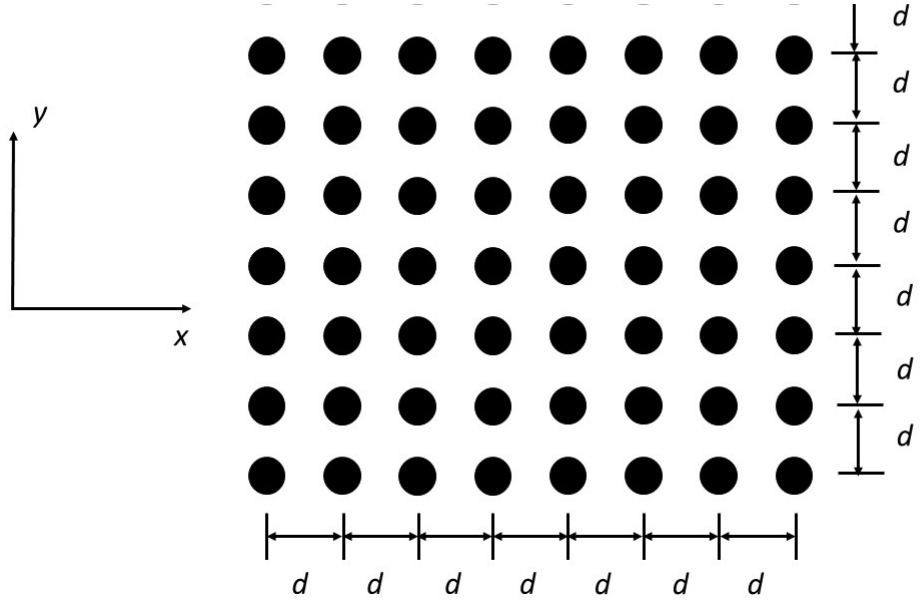

Figure 3.26: MR_PB test- Initial Conditions.

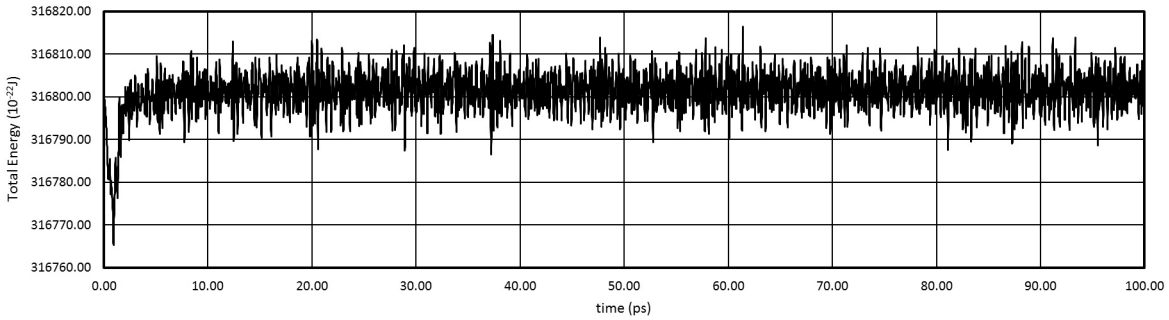

Figure 3.27: MR_PB test- Evolution of the total energy of the system as a function of time.

3.7 Efficiency Study

A MR_PB linear contact detection algorithm is consisted of an MR_PB linear sort and then an MR_PB linear search. A set of validation experiment is set up to test the

MR_PB linear contact algorithm.

The experiment consists of varying number of N particles with interspacing d along the x and y directions. For each different value of N, the CPU time for contact detection per 10 loops are solved. All results are obtained on a computer with Intel 3.3GHz random access memory (RAM) space. The results can be seen from Figure 3.29 and Figure 3.30, the number of particles are changing from $N = 7,813$ to $N = 1,998,001$. It is evident that the CPU time for MR_PB is proportional to N, as all components of the MR_PB contact detection are linear.

The results show that MR_PB sort and search algorithm are linear contact detection algorithm, and it can greatly save the computational time needed for a given simulation.

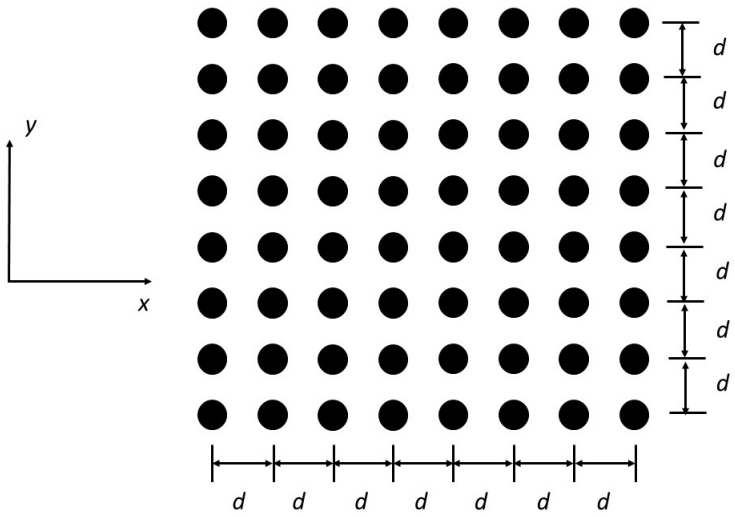

Figure 3.28: Atoms are initially placed in a $2D$ bcc configuration with interspacing of d.

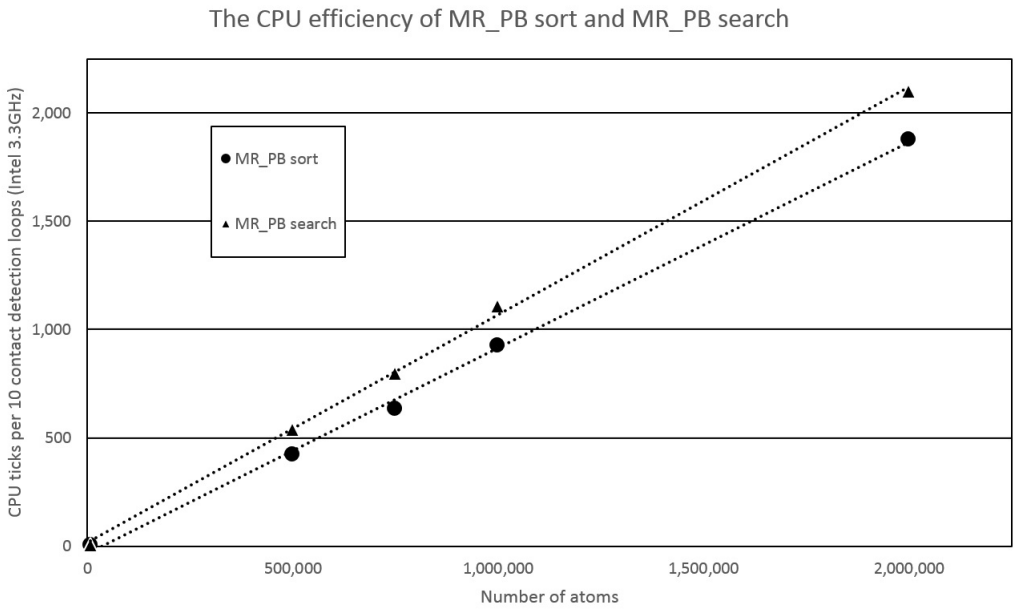

Figure 3.29: The Comparison between CPU time against the number of atoms between binary sort and MR_PB sort. The CPU time for MR_PB sort is linearly proportional to the number of atoms.

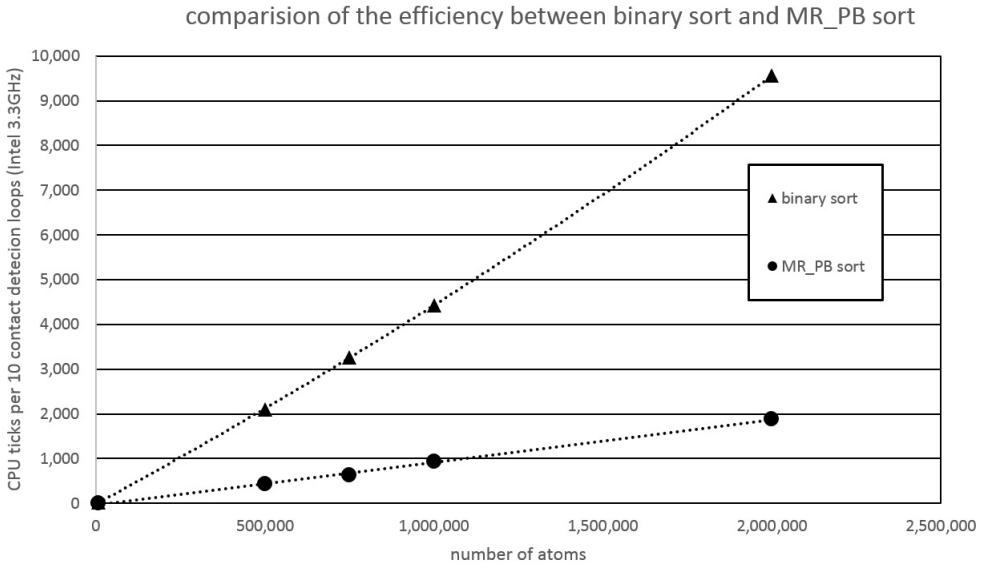

Figure 3.30: The CPU time as a function of the number of atoms in both MR_PB sort and MR_PB search algorithm. The total CPU time of the MR_PB contact detection is proportional to the number of atoms as well.

3.8 Conclusions

A verification example has been performed to test the accuracy of the MR_PB algorithm. It has shown that there's no jump in the total energy of the system and the total energy of the system is conserved, which in turn suggests that the MR_PB algorithm catches all contacting pairs and thus proves the validity of the developed the MR_PB algorithm.

Chapter 4

Shared-Memory Based Parallelization Solutions of YNANO

4.1 Introduction

As explained in Chapter 2, OpenMP is a shared-memory API that supports shared-memory multiprocessing in C++, C and Fortran.[6,25,27] In OpenMp, a master thread forks into a number (depending on how many processors there are on a single node) of slave threads, and the task is divided among all the threads. All threads run concurrently, with the runtime environment allocating threads to different processors.[23,79]

The advantage of OpenMP over MPI is at the same time its limitation, that OpenMP solutions could only be implemented on one single node. Because of its portability and ease of use, OpenMP has been actively applied to incrementally parallelize a handful of MD codes and has been applied to a range of simulations.[4,3,28,36,56,103,119,157]

In shared-memory systems, memory is accessible (visible) to all processors on the same node. The workload is shared among the cores on a single node in threads. The sequential code runs up till a point when work sharing is called in, the core that runs the sequential code becomes the master thread, it then distributes the work among the slave cores as in slave threads. Each core copies from the memory into its cache (invisible to other processors) the variables it needs to run its job. Cache stores private variables and public variables. Private variable is invisible to all other processors throughout the work-sharing, public variable is theoretically visible to all threads. However, because each threads stores variable in its private cache and public variable is updated in the memory non-deterministically, it may happen that at a given time, different core cache might have different value for the same public variable.

One way to make sure all processor has the most updated value for the same public variable is to enforce explicit synchronization point, where public variable in each individual cache updates its value in the memory which is visible to all threads.

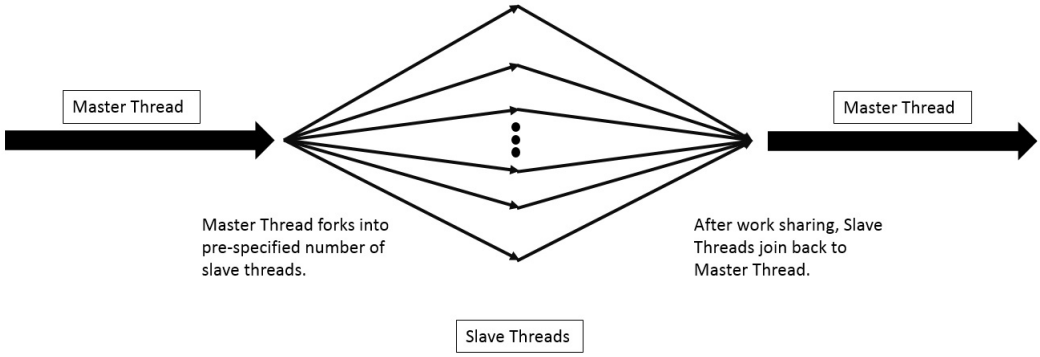

Figure 4.1: The Master Thread forks into multiple Slave Threads upon work sharing. In the end, all Slave Threads will join back into Master Thread.

4.2 Work Sharing Constructs of MR_Search Algorithm

Before parallelizing the MR detection algorithm, it is imperative to check if the MR algorithm could be parallelized at all, that is, to determine the degree of variable dependency in the algorithm.[36]

As is explained in Chapter 2, the MR contact detection consists of two parts, the MR_sort that sorts the cell list spatially, and the MR search that searches for each contact particle its neighbouring particles. In YNANO, MR search is immediately followed by the inter-distance calculation of particle pair and inter-particle potential calculation if the pair is decided to be within the contact range.

There are two possible approaches to parallelize the MR_sort, each targets at different entry point in the MR_sort routine.

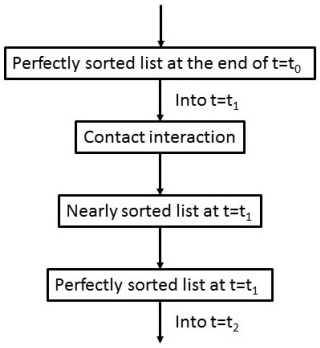

Figure 4.2: Approach one targets at the nearly sorted list at t_1 considering the position updates. Approach two targets are the nearly sorted list at t_2 without considering the position updates.

The first is to divide the current time (t_1) nearly sorted cell list into sub-lists to be sorted on individual threads. The current cell list is a nearly sorted one because after position update, the spatial order could be disturbed. For example, if the work is to be split between two cores, $C1$ and $C2$ then split the nearly sorted list L into $List_0$ and $List_1$, in which all particles in $List_0$ have smaller integerized coordinates than the particles in $List_1$. The problem with this work-sharing construct is that the data structure in the MR contact detection is a doubly connected (next pointer and previous pointer) cell list, the nearly sorted cell list does not conform to the spatial hierarchy of the ordering. Performing a spatial decomposition of the cell list will not work because the brutal-force split-up of sub-list might not have a consistent parsing among particles.

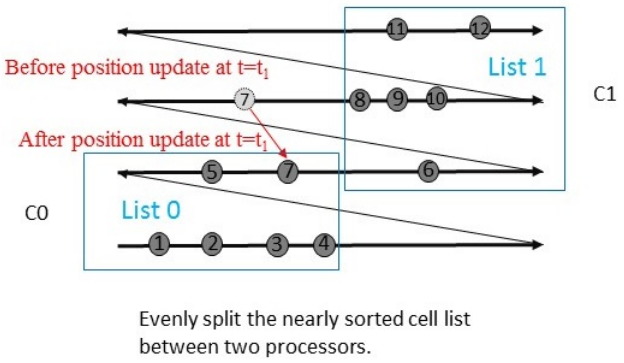

Figure 4.3: Work is shared between two processors evenly by splitting the nearly sorted list at t_1. The particles $C0$ cover have smaller integerized coordinates than $C1$.

The second is to divide the current time (t_1) nearly sorted cell list that is before the position update into sub lists to be sorted on individual threads. For example, at current time t_1, two cores $C0$ and $C1$ will split the previous perfect sorted list L into

$List_0$ and $List_1$ of equal sizes, in which all particles in $List_0$ have smaller integerized coordinates than the particles in $List_1$. This approach steers clear of problem which the first approach encounters, because the perfectly sorted list provides a consistent spatial parsing of the particles. But the problem with this approach is that, even if $List_0$ and $List_1$ are individually sorted into perfectly sorted list $List_0*$ and $List_1*$, some particles in the end of the $List_0*$ might have great integerized coordinates than some particles in the head of the $List_1*$. Should this happen, additional sort is needed to sort this transitional list region. This problem results from the fact that even though according to the stability criteria that no particle can move a greater distance than the side of a cell in any given time step, this small change of spatial position can translate into drastic change in its position inside a cell list. Therefore, more work needs to be done to concatenate the two sorted list back together, in this sense, the parallelization will not be highly efficient if the number of simulation particle is huge.

In a typical MR search work sharing routine, each threads has a list of contactor particles of size p/n, for p/n contactor particles, the contact mask is slightly bigger than p/n so that it has to extend into the previous thread's contactor particle list for an extra $n*$ particles. And since inter-particle force is calculated only once for each pair, the inter-particle net force on the $n*$ particles might be different between two adjacent threads. *# pragma omp critical* block is introduced to ensure that the velocity (public variable) is accessed by one thread at a time.

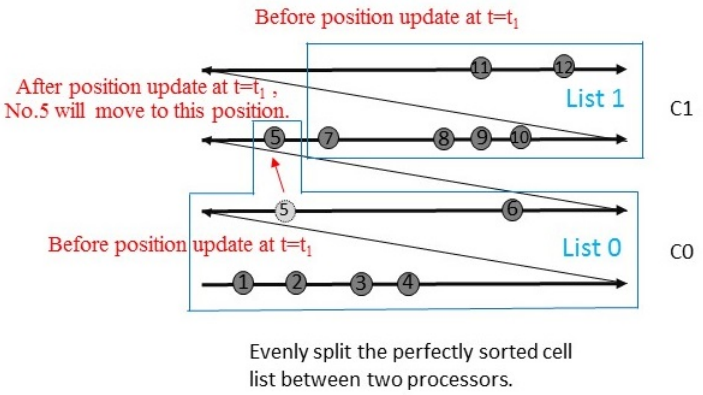

Figure 4.4: Work is shared between two processors evenly by splitting the perfectly sorted cell list at t_1. The particles $C0$ cover have smaller integerized coordinates than $C1$.

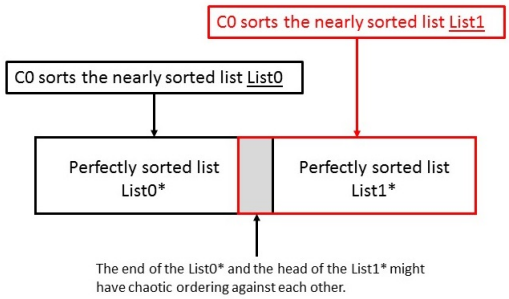

Figure 4.5: Within each perfectly sorted list in C_0 and C_1, it could happen that some particles in the end of the $list0^*$ have great integerized coordinates than some particles in the head of the $list1^*$. Should this happen, additional sort is needed to sort this transitional list region.

To summarize, it is inefficient to parallelize the MR Sort Algorithm using work sharing constructs.

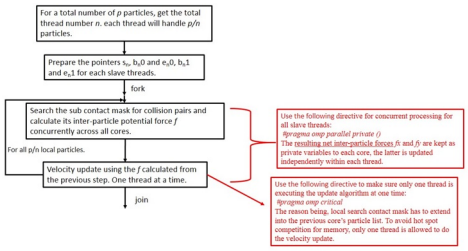

Figure 4.6: The work flow for the parallelized MR search routine using OpenMP.

#pragma omp critical block makes sure that at any given time, only one thread is allowed to execute this block of code. Employing critical block at the velocity update stage makes sure that there's no hot spot for memory competition for public variables, and also, all public variables are updated across the processors.

The critical block doesn't harm the efficiency of the work-sharing greatly, because as can be seen from the analysis above, force calculation and distance calculation take up the majority of the CPU time.

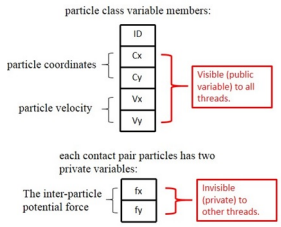

Figure 4.7: Inside the work-sharing of the MR search routine, public variables are the particle coordinates, private variables are the force components calculated within each processor.

The MR_search in YNANO is implemented to do three things sequentially. For each contact particle,

- Firstly, its contact mask is parsed to pick out all potential neighbouring particles that might in contact with it.
- Then, the contact particle is measured against each of its potential neighbouring particles for the inter-particle distance between them.
- If the distance is smaller than the collision radius, the particle pair is considered to be in collision. The inter-particle potential force will be calculated, and subsequently the acceleration on the particle pair will be updated.

In YNANO, the above three steps are the most time-consuming part in the code, because it involves complicated calculation of square and square root. It has been tested that in most commercial software, contact calculation takes up to 90% of all CPU time.

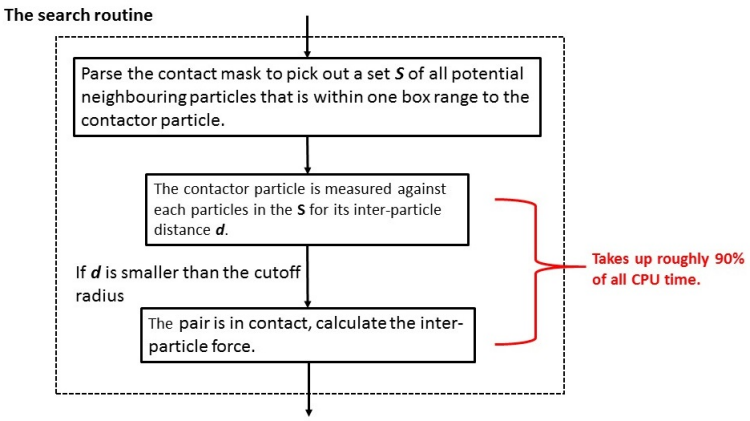

Figure 4.8: Inside the search routine, inter-particle distance calculation and inter-particle potential force calculation take up 90% of all CPU time in the whole simulation.

The MR_search is parallelizable because

- Firstly, the process of parsing through the contact mask and marking the beginning/ending pointers is independent for each contact particle.
- And secondly, only particle coordinates is required to determine whether or not a particle pair is in collision, and at this stage particle position is not subjective to change, therefore each thread will only be accessing static data.

In other words, the MR contact search is parallelizable because the information (particle position) needed for calculation doesn't changed during the work sharing routine. For the ease of convenience, the following section will illustrate on a two-core case in $2D$, but it should be noted that the parallelization strategy is a general purpose one and it could be easily adapted into an arbitrary number of cores provided they share the same memory, and as well as into $3D$.

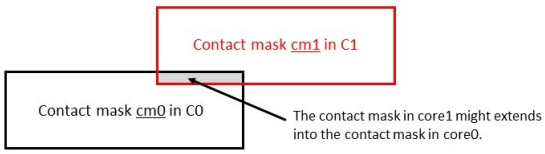

Figure 4.9: Because contact mask covers the particles that are within one box length around the contactor particle and with a smaller integerized coordinates than the contactor particle, therefore, the contact mask in core1 needs to check against the positions of some particles that are the contactor particles in *core*0.

Because contact mask covers the particles that are within one box length around the contactor particle and with a smaller integerized coordinates than the contactor particle, therefore, the contact mask in core1 needs to check the positions of some particles that are the contactor particles in *core*0.

In Figure 4.3 and Figure 4.4, the contact search of a cell list with 30 particles is shared among 2 cores, each core will cover 15 particles. It should be noted that number of particle is its cell-list ordering number.

As explain in Chapter 2, a sequential $2D$ MR Search algorithm needs three categories of pointer for mask parsing.

- A working pointer S that points to the contact particle.
- A set of two pointers pointing to the particles in the contact mask that has the same integerized y coordinate as the contact particle, they are denoted as b_1 (begin) and e_1 (end).

- A set of two pointers pointing to particles in the contact mask that has one integerized y coordinate smaller than the contact particle, they are denoted as b_2 and e_2. in the beginning of a sequential code, all 5 pointers start from the list head, and they uni-directionally progress toward the list tail.

Upon parallelizing the sequential 2D MR contact search, different cores cover different segments of the cell list, therefore, the three categories of pointers are placed at different positions for each core. For example in Figure 4.10 for $C0$, it covers the first half (with smaller integerized coordinates) of the cell list. As long as $C0$ is concerned, things operate as usual, all its three categories of pointers ($S0$, $b_1 0$, $e_1 0$, $b_2 0$ and $e_2 0$) will start from the list head and parse towards the middle.

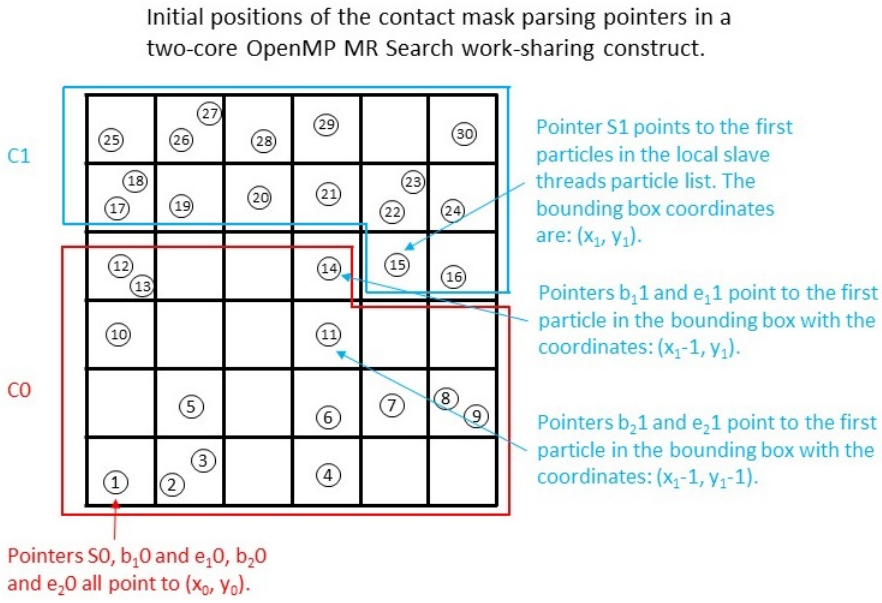

Figure 4.10: For $C0$, $S0$, $b_1 0$ and $b_2 1$ will point to particle $No.1$. For $C1$, $S1$ will point to particle $No.15$, $b_1 1$ will point to particle $No.14$, and $b_2 1$ will point to particle $No.11$.

$C1$ takes over the second half (with larger integerized coordinates) of the cell list, it is obvious that its working pointer $S1$ will point to particle $No.15$. The second and third set of pointers will invite more careful considerations. $b_1 1$ should point to the first particle in the immediately left cell of the contact particle, that is particle No. 14. By the same principle, $b_2 1$ will point to the first particle that is in the box which is immediately under particle No.14, which is particle No.11. Pointers $b_1 1$ and $b_2 1$ extend into the first half of the cell list which is processed by the first core, this will

not raise a problem because only the workload of parsing the contact particle is split among the cores, which is performed by the first $S1$ pointer.

It should be noted that under the current shared-memory based work sharing constructs, each core gets a same copy of the algorithm and they execute the same code. Different initialization of the pointers needs to be assigned to different threads before the master thread forks.

In this case, pointers S, b_1, e_1, b_2 and e_2 need to be initialized differently for individual processors. In the example provided above, $C0$ is initialized with $S0$, b_10, e_10, b_20 and e_20, and $C1$ are initialized with $S1$, b_11, e_11, b_21 and e_21.

4.3 Validation

To test the developed shared-memory based parallelization solutions of YNANO, a test example of a box filled with $250,000$ gaseous Argon atoms is set up. The initial temperature for the whole system is set at $120K$, the number density of the system is $9.612E-3\,/\mathring{A}^2$, the initial interspacing between particles is $10\mathring{A}$. Tests are run on 1 core (sequential), 2 cores and 8 cores respectively, system total kinetic energy and efficiency are compared as follows.

number of processors [-]	CPU time[-]	speedup[-]
1	276	1
2	171	1.48
8	41	6.17

Table 4.1: Calculated speedup for a box filled with $250,000$ gaseous Argon atoms on sequential code, paralleled code with 2 processors and parallelized code with 8 processors.

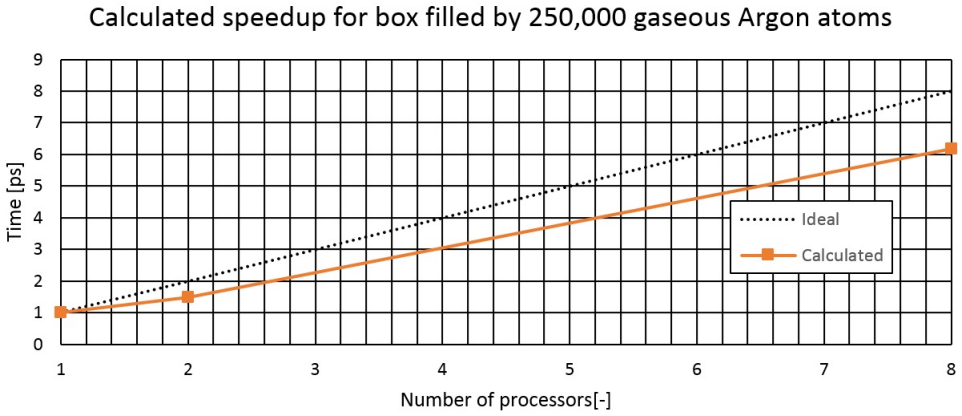

Figure 4.11: Calculated speedup for box filled up 250,000 gaseous Argon atoms.

Results showed that the speedup is close to linear. The discrepancy between the calculated speedup and the ideal speedup could be due to the fact that OpenMP is only partly parallelized. The performance of 8 core is significantly better than that of the 2 cores, as overhead (from cache to main memory) could be hugely expensive, the observed huge speedup could most probably result from the specific implementation of the memory management details in the cluster node.

Chapter 5

NOVEL SPATIAL DECOMPOSITION BASED PARALLEL SOLUTIONS FOR MOLECULAR DYNAMICS SIMULATIONS

5.1 Introduction

In the context of constant and fast progresses in nano technology, discontinua based computation simulations are becoming increasingly important, especially in the context of virtual experimentations. However, the efficiency of discontinua based nano-scale simulations is still greatly limited by the capacity of CPU (the number of simulation particles in the system).[1,2,6,10,11,12]

It is a widely accepted view that parallelization will play an important role in solving this problem. In this thesis, two parallelization approaches have been undertaken to parallelize the YNANO discontinua simulations. In this chapter, parallelization solutions of the YNANO on the distributed-memory systems using MPI (Message Passing Interface) is developed, the design and implementation of which are described in detail. The developed MPI parallelization solutions are built upon the original MR linear contact detection algorithm which is implemented in the sequential YNANO, the developed solutions preserves the linearity of both MR_sort and MR_search algorithm.

5.2 Comparison between a sequential and a parallel YNANO code

As discussed in Chapter 2, a regular molecular dynamics simulation is composed of four steps, read in input, generate particle initial velocity and position, contact detection and interaction, and finally, update particle velocity and position. Upon parallelization, the procedures of a parallelized molecular dynamics is outlined in Figure 5.1, in which, two more steps are included on top of the sequential steps. These two steps are outlined with dashed lines, namely, the force transmission message passing and the particle migration message passing.

Force transmission happens among adjacent processors right after contact detection and force calculation, the aim of the transmission is to share with each other (processors) the interaction force acted on the interfacial particles that are shared among those processors, because interfacial particle might be subject to the interaction forces exerted by internal particle within each processor's local domain which is hidden from the other processors. Force transmission happens at every time step whenever there are shared interfacial particles.

Particle migration happens just before the end of each time step. The reason why particle migration is necessary is because when particles move outside of one spatial simulation sub-domain and enters another, they need to be added into the workload of the processor of interest. Recall that in distributed-memory systems that each processors work independently (memory is not shared) from each other, the processor of interest is ignorant of any particle attempting to enter its sub-domain, migration serves the purpose of transmitting particle data across processors. Since communication overhead cost is very expensive, judgement of migration is very crucial in performance streamlining.

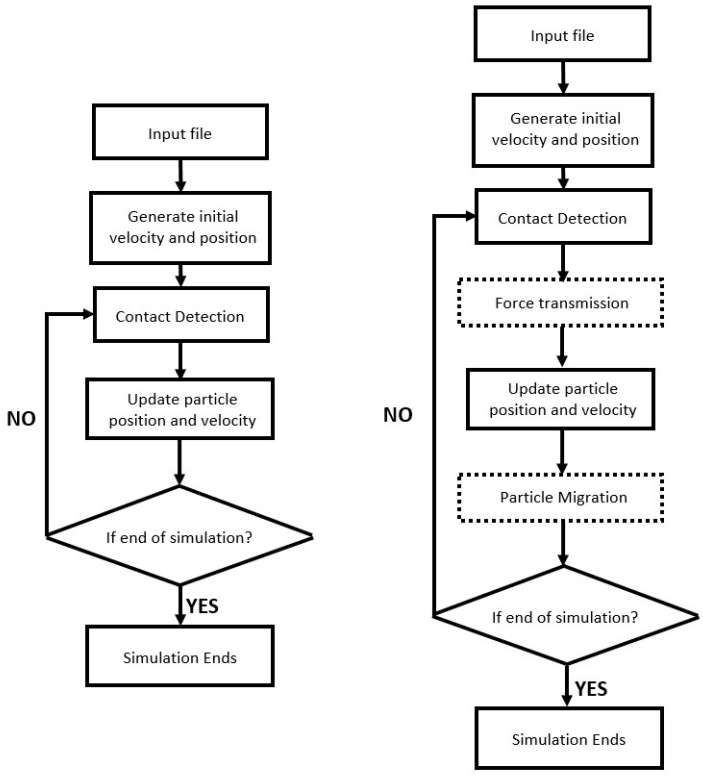

Figure 5.1: The comparison between a sequential molecular dynamics simulation and a parallelized version. Additional steps are outlined with dashed lines.

5.3 Domain decomposition

In a sequential code, the calculation domain is the whole domain, however in a parallelization MPI code, different processors are handling different parts of the whole domains, therefore the whole domain needs to be broken up into smaller sub domains, and each processor only handles the particles residing within its sub domain. For a parallelization solution without dynamic load balancing, it is advisable to break the domain into equal sizes with roughly the same number of computing particles. Roughly equal workload is advisable not only because it fully utilizes the computing capacities of all processors, but also in the context of communication, if certain processors take much longer time to reach the communication checkpoint (message passing) than others, the other processors will need to sit in idle and wait for those heavily burdened processors.

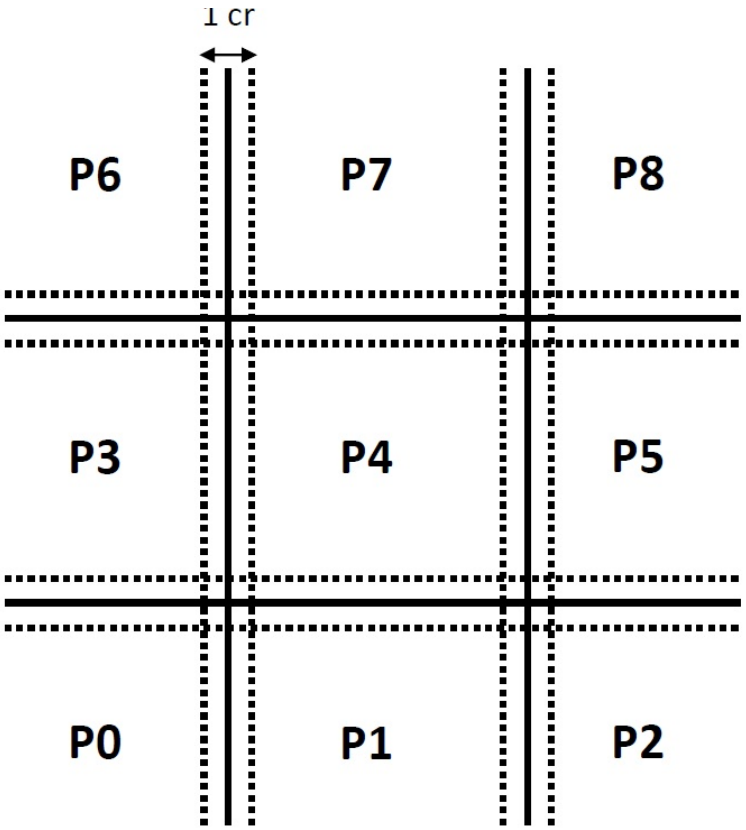

Figure 5.2: The size of the buffer zone is 1 cutoff radius of the particle used in the simulation.

In this thesis, the whole rectangular domain is geometrically divided into smaller rectangular sub-domains according to pre-specified number of processors. Also, since the simulation problem is assumed to be isotropic, the smaller sub-domains are of the same sizes. Each processor will handle only the particles in the smaller sub-domain assigned to it, and plus the interfacial particles in the outer buffer zone around it. The buffer zone particles are interfacial particles that have a copy of itself in all adjacent processors, together with the internal particles that only one copy of it exists across all processors, altogether they provide a complete picture for the particle contact interaction for each sub-domain.

The size of the buffer zone is determined according to the force field model of the molecular dynamics simulation. The contact model for the molecular dynamics is the L_J $12-6$ potential field, the cutoff radius for the Argon atom is 2.5σ, therefore, the buffer zone should be at least as big as the cutoff radius. It should be noted that spatial decomposition is possible without constructing the buffer zone. The reason why buffer zone is constructed here is to facilitate the MR linear contact detection algorithm. The

following section will compare the buffer zone free design against buffer zone design.

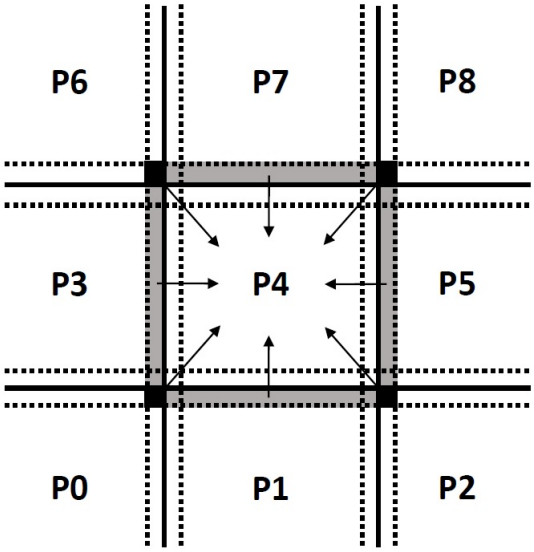

Figure 5.3: Without the buffer zone scheme, P_0, P_1, P_2, P_3, P_5, P_6, P_7 and P_8 will all need to directly send to P_4.

Without the buffer zone, two problems will arise that centre around the Contact Detection and Interaction stage.

1. Huge communication overhead: A typical processor will receive position information from 8 neighbouring processors for the particles lying within 1 buffer zone length (cut-off radius).

2. Huge communication redundancy at every time step: After receiving those messages, the local processor will need to sort all these 'alien particles' into the local particles list to form a perfectly sorted list for the MR contact search algorithm. The problem is, between each time step, the message and the extended particle list are not saved (if they are, they would be similar to the buffer zone model), and therefore, there exists huge redundancy in communication.

In contrast, with a buffer zone, the local processor already has within it a complete copy of all the interfacial particles that would be in contact with its internal particles, therefore an MR contact list is naturally set up and requires no modification to the MR contact algorithm, also the end product will preserve the linear efficiency processing time proportional to the number of particles.

As can be seen, no message passing across processors is needed during contact detection, however after contact, each processor needs to transmit and update the net force of its interfacial particles with its neighbours. Also, under current cell list design

which sorts particles with a Cartesian precedence of y over x, it is easier to merge a few newly migrated particles into the existing perfectly-sorted list than to sort the 5 jumble sections of particles.

The migration process is also different for MPI with buffer zone and without. For the former case, each processor only needs to pass messages twice, horizontally and vertically, and the end product will cover the diagonal case. However in the latter case, each processor will need to pass messages in two additional diagonal directions as well. The reason why this difference is present lies in the fact that in the former scheme, adjacent processors share the adjacent interfacial particles, therefore for previously internal particles that travel diagonally, they are copied into all adjacent processors, however for the latter case, each particle will have one and only copy across all processors as seen in Figure 3.11.

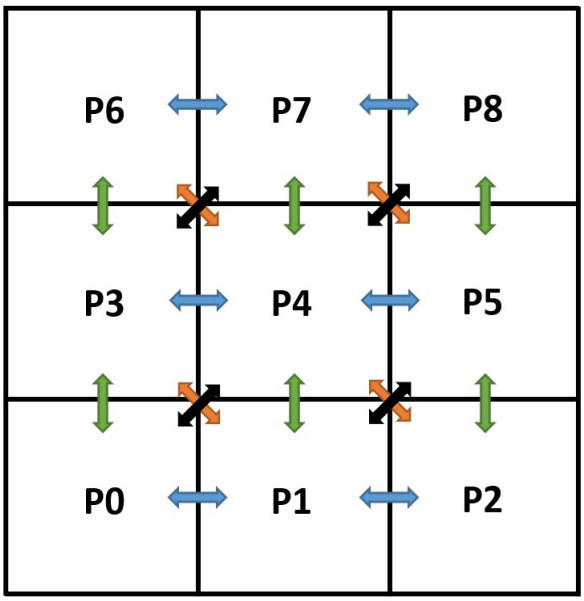

Figure 5.4: The message passing scheme without buffer zone. Besides the horizontal and vertical transmission, diagonal transmission is also required.

It is evident from the discussion above that the difference between domain decomposition with buffer zone and without is the MR_sort and search efficiency, it arises because the former retains the efficiency of MR_sort and search, and the latter requires additional message passing during the migration stage, not to mention that opening up a channel is very expensive, therefore in this thesis, the buffer zone scheme is adopted.

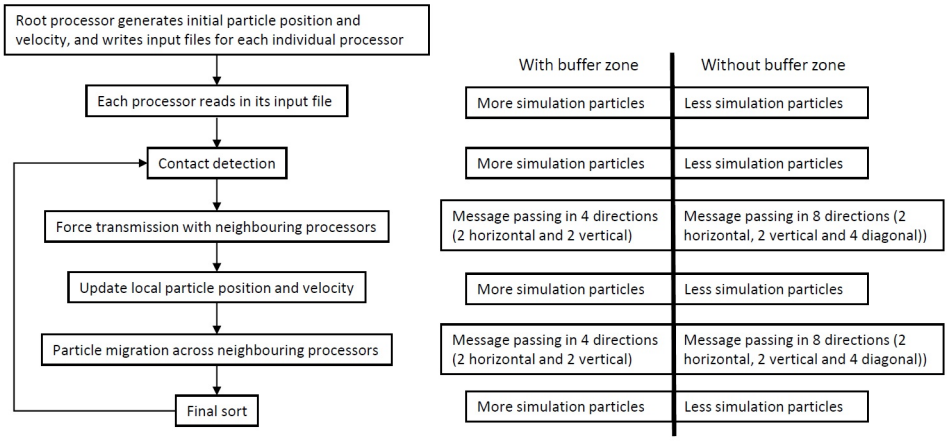

Figure 5.5: Comparison between simulation with buffer zone and without.

5.3.1 Classification of particles according to their position in the sub-domain

After domain decomposition is performed, all particles are divided into different categories according to their position within the sub-domain. If a particle is located totally inside one processor, then it is marked as I (internal) particle. All other particles are interfacial particles that are shared by two or more processors, they can be further divided into three categories:

1. Section A and section C include particles that are shared with the local processor's horizontal neighbours. Each particle in section A and section C has two copies across all processors.

2. Section B and section D include particles that are shared with the local processor's vertical neighbours. Each particle in section B and section D has two copies across all processors.

3. Section AB/BC/CD/AD (the corner sections) include particles shared by four neighbouring processors around the corner. Each particle in the corner section has four copies across all processors.

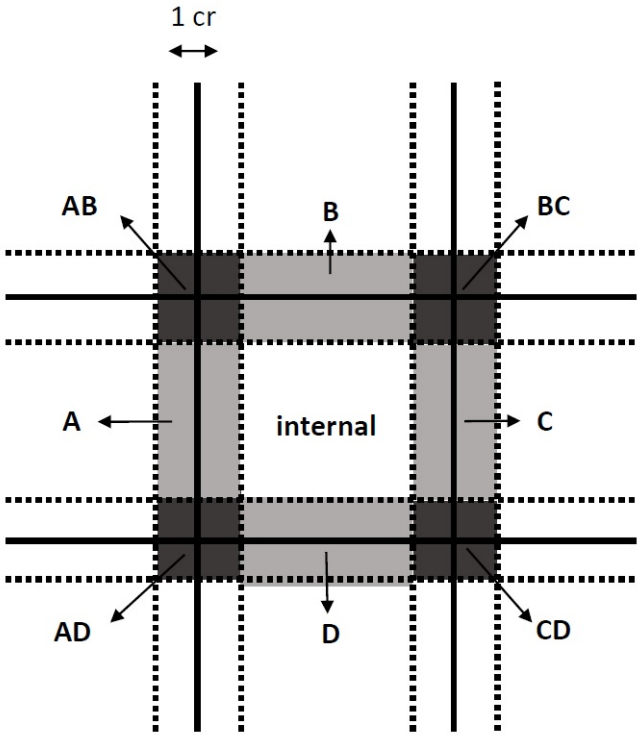

Figure 5.6: Different sections of a sub-domain according to the spatial position.

This categorization of particle position is necessary because processors need to communicate with each other on their shared interfacial particles, and it helps to have a book-keeping system to label those interfacial particles. During the stage of force transmission and particle migration communication, as processors pair up for communication, each processor will simply call in the list of shared interfacial particle instead of parsing the whole sub-domain every time. This procedure will be further explained in the following sections. Upon classification, an integer flag is assigned to each section. The reason why an integer flag is necessary will be explained in the force transmission section.

5.4 Memory Management

The original Ynano is written in C++. The sequential Ynano deals with a fixed number of particles, therefore the memory it requires is constant. However upon parallelization, each processor will only spatially handle a portion of the total particles, it is inevitable that for any local processor, at a certain time, some particles will leave its original domain and some will enter it, and net flux will not be zero. Therefore, each

processor will require flexible amount of memory at each time step, and memory management in C++ could be a real problem (such as memory leak and stackoverflow) if not handled properly.

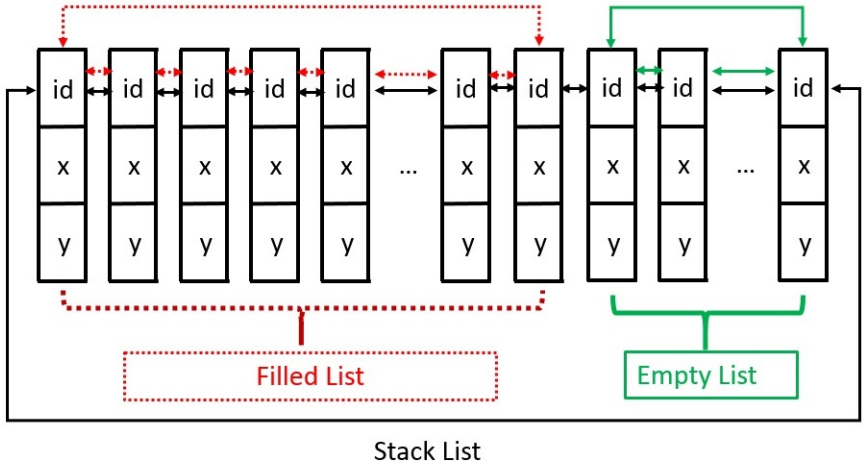

Figure 5.7: In the beginning of a simulation, the Stack List is entirely composed to mutually exclusive Filled List and Empty List.

To deal with this delicate memory issue, a base list called Stack List will be constructed for every processor at the beginning of the simulation, this list will be allocated $r = 120\%$ of the memory it actually requires (the actual number of particles it is assigned to upon initialization). For example, if $n = 1000$ particles are divided into 4 processors, each processor will handle $n_0 = 250$ intrinsic particles, in the end, the Stack List will be allocated with a memory size proportional to 300 particles , allowing migration fluctuations.

$$n_0 = (n \div p) \times r$$

The Stack List is composed of 2 mutually exclusive lists, the Filled List and the Empty List. The Filled List houses the all the local particles, the Empty List documents all immigrating particles. When there are only emigrating particles, the Filled List shrinks, and the Empty List expands. When there are immigrating particles, the Empty List has to cough up some space for those new immigrating particles, these new immigrants then form a new temporal list called the NewP List within the Stack List.

At the end of each time step, should there be a NewP List, the NewP List will first be incorporated into the Filled List, then sorted into the local perfectly sorted particle list—for contact detection and interaction in the next time step.

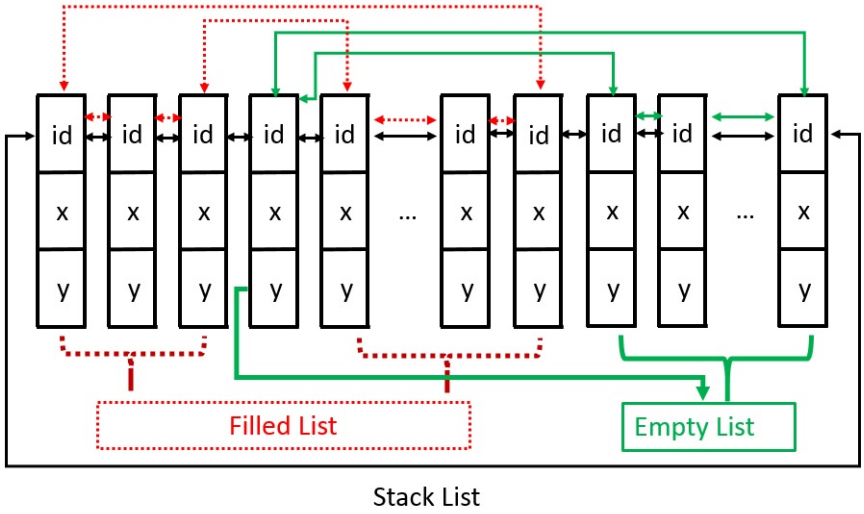

Figure 5.8: When particles leaves the local processor's sub-domain, the local Stack List is still entirely composed to mutually exclusive Filled List and Empty List, but the Filled List shrinks for the Empty List to expand.

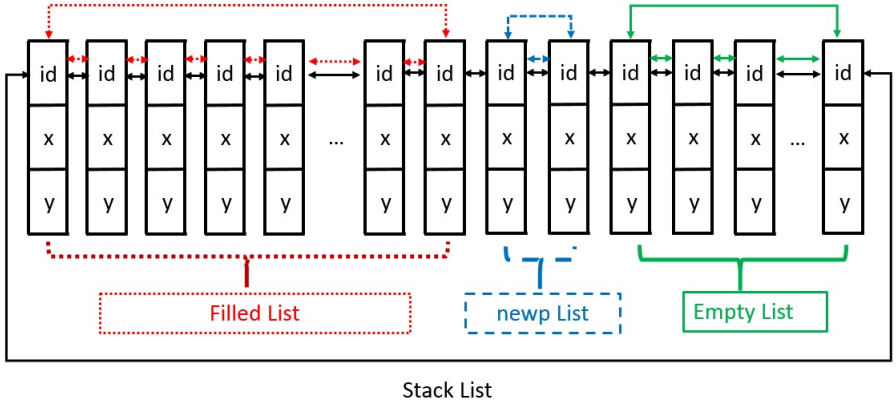

Figure 5.9: When new particles immigrate into the local subdomain, a new list is created, the NewP List.

The Empty List has a very important role in memory management. During migration, if a local process decides that one of its local particle has moved outside of the domain, three things will happen:

1. In the Stack List, the class object that stores the position and velocity of this particle will be cleared of content.

2. This class object will be cut off from the Filled List, this is done by cutting off its bi-directional links in the Filled List, and concatenating its previous object and next object together.

3. This class object will be added into the Empty List.

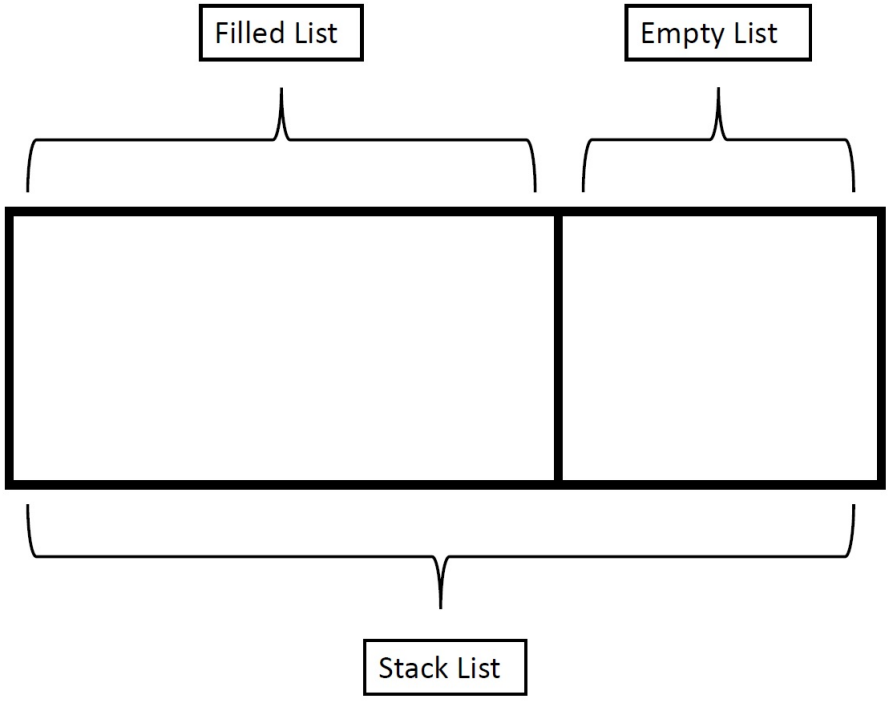

Figure 5.10: In the beginning, there are only Filled List and Empty List.

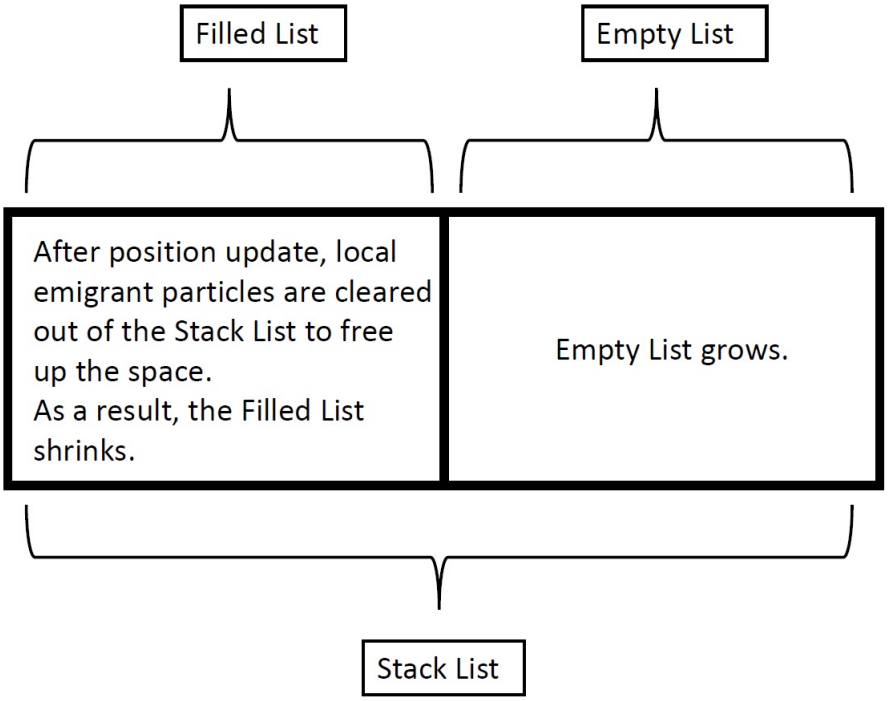

Figure 5.11: Filled List shrinks, Empty List expands.

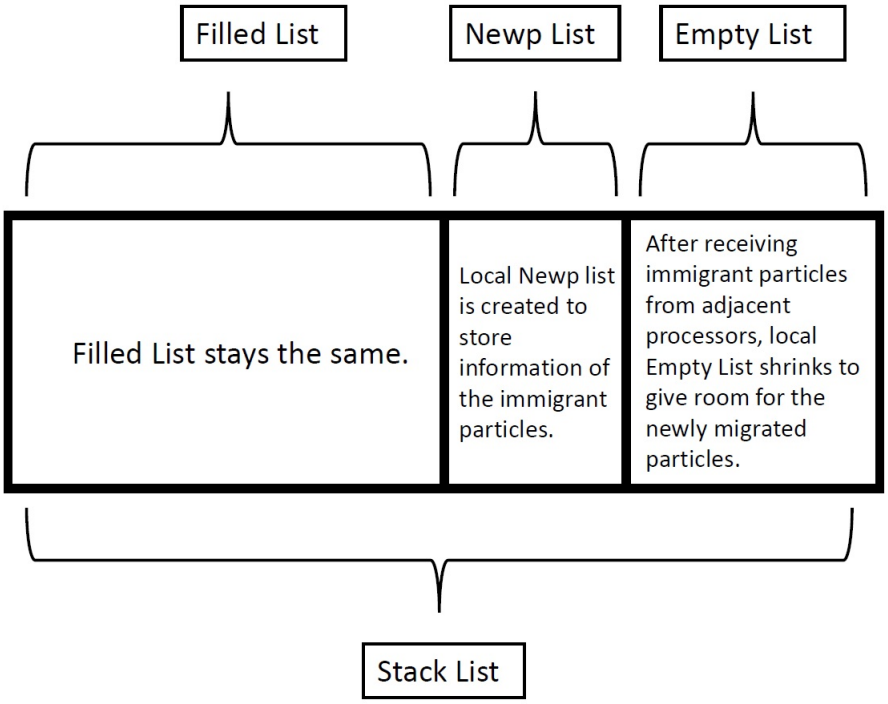

Figure 5.12: Filled List stays the same, Empty List shrinks to make room for the NewP List.

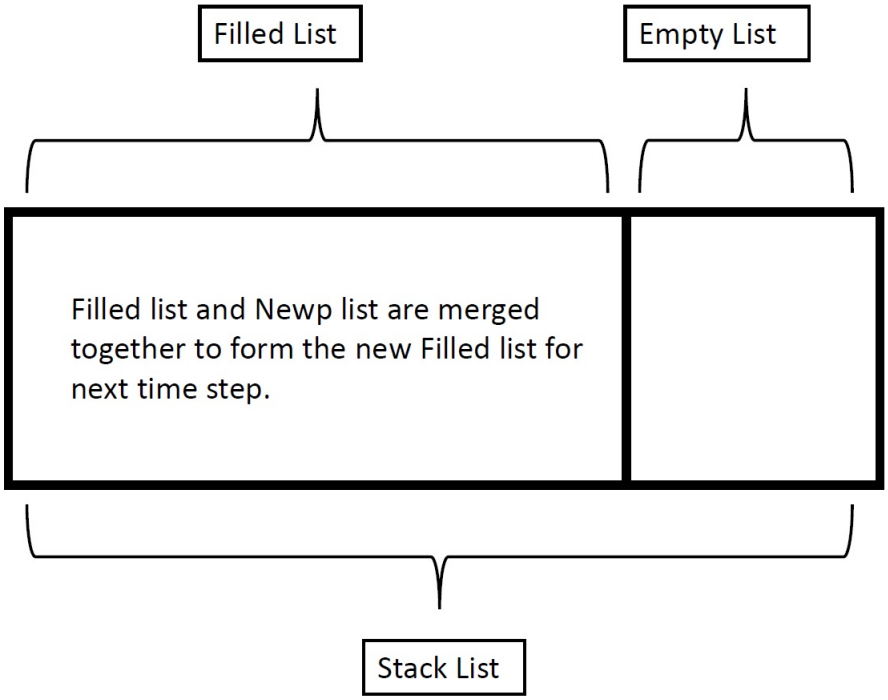

Figure 5.13: The NewP List is merged into the Filled List.

It should be noted that throughout the operations, the size and connection of the Stack List remains constant.

Likewise, if a new particle is received into the local processor, three things will happen:

1. The Empty List will need to cut one of its item (an object) out of its list.
2. This object will be added into the Filled List.
3. The position and velocity of the newly added particle will be written into the object.

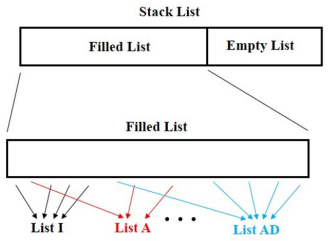

Figure 5.14: Each spatial section (I, A, B, C, D, AB, BC, CD, AD) in the local domain has a sub list on top of the Filled List.

Also, the data structure has three levels. The Filled List is the list that MR_sort operates on. The Filled List is also the backbone of the section lists in the domain. The reasons why sub lists are created is:

1. Better bookkeeping during force transmission.
2. More straightforward judgement process in particle migration.

5.5 Parallelization of Contact Detection and Interaction

5.5.1 Contact Detection

Contact detection is the stage at which all local particles are checked against its nearest neighbours (lying within one bounding box length) for contact, and in the case where the inter-particle distance is smaller than the cutoff radius, interaction force will be calculated subsequently in the contact interaction stage according to the chosen force field model (the LJ $12-6$ in this thesis).

In the sequential Ynano, MR_sort algorithm sorts all the particles and MR_search algorithm searches for potential contact pairs, both algorithm have a linear processing time proportional to the number of particles. In the parallelized Ynano, the MR_search is kept without modification, and the MR_sort is modified to accommodate the possible newly migrated particles, the details of the modified MR_sort will be explained in detail in the migration section.

Throughout the contact interaction process, the Filled List is used as the contact detection list. At the first time step, the Filled List is sorted in a binary fashion and then fed into the MR_search algorithm. Details of the MR_search algorithm can be found in the previous chapters.

After contact interaction, the perfectly-sorted Filled List (defecto the MR list) dips into near-sortedness. After particle migration, some previously local particle have been cleared out of the MR list, some new particles have been added into the NewP List. The new particles inside the NewP List are linked into different section sub-lists, noted that there is no ordering within the sub-lists in the NewP List. The following needs to be done to finely incorporate the newly immigrating particles into the local particle list system:

1. To corporate the Filled List and NewP List together, into a Filled* List. This could be done without sorting. Simple appendage is needed.

2. To sort the Filled* List into a new MR* List, preferably linearly, for contact detection in the next time step.

3. To corporate the respective section sub-lists in Filled List and NewP List together. To sort the resulting section sub-lists, preferably linearly, for efficient message composing in the next time step.

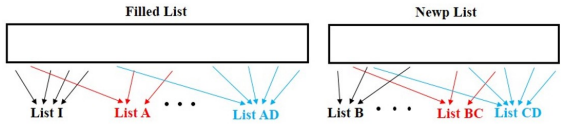

Figure 5.15: For both the Filled List and the NewP List, section sub-lists are created. The aim of the sort is of two levels, first, to combine and sort the Filled List (MR list) and the NewP List; second, each respective section sub-lists are combined and sorted as well.

To accomplish step 2, both the Filled List and NewP List need to be perfectly sorted internally.

The Filled List has already been sorted from the MR_sort in the previous steps with linear time efficiency, out of simplicity considerations, the Filled List will be referred to as the MR List onward. Even though particle migration may have taken some particles off the Filled List, the spatial ordering of the MR list is not disturbed.

The NewP List will not be able to use MR_sort because no previous integerized coordinates information is available, generic sorting techniques will need to be applied.

First, both lists are sorted internally. Then, start parsing the two list from the list head at the same time. If the size of the MR List is zero, which means there's no local particles, then no further parsing is necessary, and all of the sorted NewP List could be safely appended to the end of the MR List. If the size of the NewP List is zero, which means there's no influx particles, then no further parsing is necessary. If the above two statements are false, then judge if the current particle in MR List is no smaller than the current particle in the NewP List, if yes, then move on to the next particle in the MR List; if no, insert the current NewP List particle into the linkage of the MR List, and move on to the next particle in the Newplist.

The algorithm described above has linear time efficiency, because the two individual parsing pointers parses its respective list only once in one direction.

The combine and sort of the section sub list is similar to the combine and sort of MR list described above.

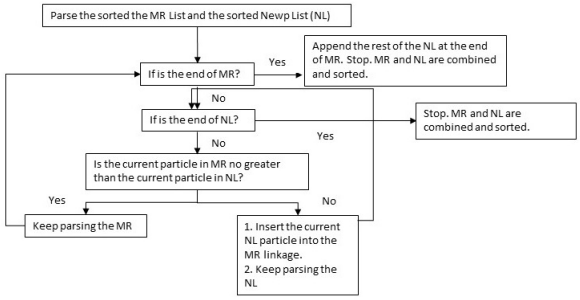

Figure 5.16: Linear sorting of the Filled List and the NewP List.

5.5.2 Contact Interaction

In a sequential code, contact interaction between particle pairs is calculated from the inter-particle force field model. In a parallelized code, the whole simulation domain is broken up into sub-domains which are assigned to different processors, and each processor has only a partial picture of the force condition of its interfacial particles. As explained above, communication across adjacent processor serves the purpose of supplying the missing information to make sure each processor is handling its local particles with sufficient knowledge.

It is to this end that the inter-particle force in the parallelized code is not as straightforward as that in a sequential code. The operation performed on the inter-particle force depends on the communication modes and communication strategy, which will be explained in the following section.

5.6 Parallelization of Force Transmission

Force transmission is the stage where information of interaction force of the interfacial particles are shared across adjacent processors. The reason why this operation is necessary lies in the fact that MPI parallelized code breaks the calculation domain among processors, that is to say there's no way for each local processors to know the full contact profile of the interfacial particles, and therefore adjacent processors need to communicate among each other to provide the piece of contact information of the shared interfacial particles that others cannot 'see'.

Force transmission consists of four steps:

1. Local processor prepares the messages to be sent horizontally.
2. Local processor receives the horizontally transmitted message, and update the interfacial particles in question accordingly.
3. Local processor prepares the messages to be sent vertically.
4. Local processor receives the vertically transmitted message, and update the interfacial particles in question accordingly.

The reason why diagonal message passing is not -necessary is because of the update in step 2. For example in fig, atom x and atom y is a unique contact pair in P_0, which means the interaction force (f_x and f_y) acting on atom y is invisible to all other three processors that share it, therefore the interaction force needs to be transmitted to all other three processors (P_1, P_2, P_3).

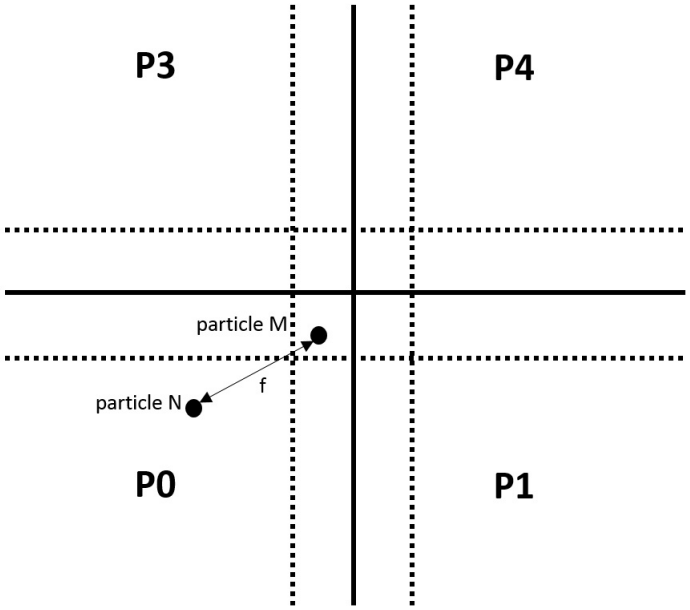

Figure 5.17: Particle M and particle N are a collision pair in P_0. Particle N is an internal particle to P_0, particle M is shared among all four processor.

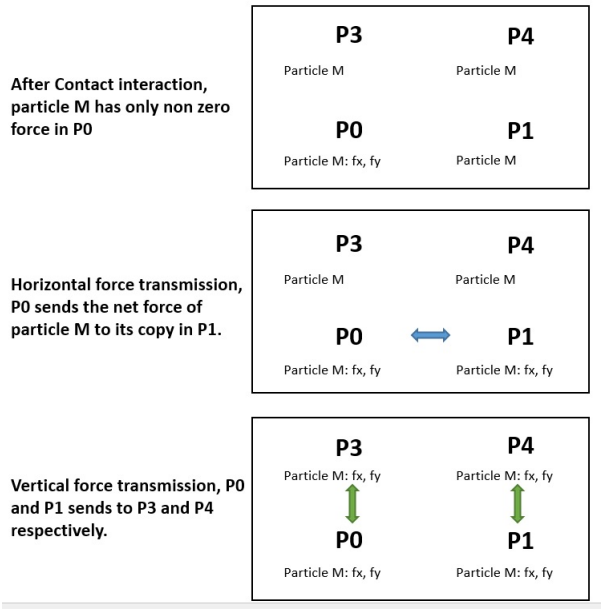

Figure 5.18: Horizontal and vertical force transmission pass the force acting on particle M in P_0 to all other three processors.

For the convenience of illustration, suppose atom y is not in contact with any other particles. First, P_0 horizontally passes the f_x and f_y to P_1. Upon receiving the message, P_1 updates the local copy of atom y from $F_x = 0$ and $F_y = 0$ with the new addition of $F_x = f_x$ and $F_y = f_y$. Then P_0 and P_1 vertically pass the f_x and f_y to P_2 and P_3 respectively. In the end, all four processors have the same force profile of the contact pair in P_0, and on all four processors atom y should have the same value of $2D$ force components (F_x and F_y).

5.6.1 Force Rescale

Under the abovementioned transmission scheme, the interaction force for interfacial particle pair needs to be rescaled. Because the interfacial particles are shared among adjacent processors, rudimentary addition of the calculated interaction force across processors could lead to repetition.

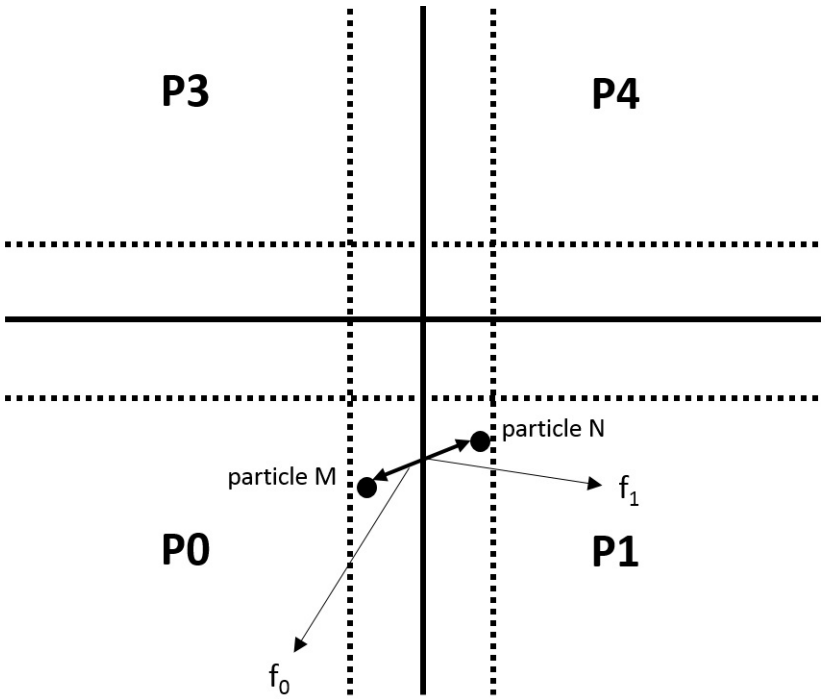

Figure 5.19: Particle M and particle N are both interfacial particles shared by P_0 and P_1. In P_0, the resultant inter-particle force between the pair is f_0, in P_1, the resultant inter-particle force between the pair is f_1. $f_0 = f_1$

For example, in Figure 5.19 particle x and particle y are both shared by P_0 and P_1. From the point of view of P_0, the pair produces f_0, and from the point of view of P_1 the pair produces f_1, $f_0$is equal to f_1. After message passing, f_1 will be added to f_0 on P_0 and f_0 will be added to f_1 on P_1, the end numerical result is $2f_1$ and $2f_0$ on P_1 and P_0 respectively, doubling the actual value. To avoid this, a scale factor is called into existence. A theoretical analysis of the scale factor is presented as follows:

Given a local particle pair, particle X and particle Y, two non-empty sets $X\{P_i\}$ and $Y\{P_j\}$ are associated to them respectively, in which P_i and P_j are the rank of processors that retains a copy of particle X or particle Y respectively. There can be at most four items in each set. An integer counter c is initialized as 0. Each item in set $X\{P_i\}$ is compared against set $Y\{P_j\}$, if $P_i = P_j$, counter c is increased by 1. In the end, the counter is the scale factor.

All possible contact pairs can be listed as follows:

Particle1 flag	Particle2 flag	scale factor	sum
1	1	1	2
1	2	1	3
1	3	1	4
1	4	1	5
2	2	2	4
2	3	1	5
2	4	2	6
3	3	2	6
3	4	2	7
4	4	4	8

Table 5.1: Classification of flags according to spatial position.

Algorithm 5.1 Load the horizontal send array

1: **integer** $flag1, flag2$ ▷ 2D velocity acceleration component
2: **integer** sum ▷ 2D velocity component
3: **integer** $ScaleFactor$ ▷ 2D position component
4: $ScaleFactor = 1$ ▷ update the velocity component for next time step
5: **if** $sum = 4$ or $sum = 6$ or $sum = 7$ **then**
6: $ScaleFactor = 2$
7: **else if** $sum = 8$ **then**
8: $ScaleFactor = 4$
9: **else if** $flag1 = 1$ or $flag2 = 1$ **then**
10: $ScaleFactor = 1$
11: **end if**

5.6.2 Message Preparation

Before sending messages, it could greatly enhance the parsing and updating efficiency if both the sending and the receiving processors agree on certain conventions about the message format.

1. The send array is written in a fixed pattern according to their position in the sub-domain.

2. Upon receiving the incoming message, the local recipient processor updates its local particles in the fashion the message is written.

Position in the sub-domain	sending direction	message content sequence (section)
East	Left	A, AB, AD
West	Right	C, BC, CD
North	Up	B, AB, BC
South	Down	D, AD, CD

Table 5.2: Message preparation sequence according to spatial position.

In this thesis, a force transmission message is a $1D$ array composed of three sections, each section includes particles in a specific region of the sub-domain. For example, a horizontal message that sends eastwards is composed of particles in section C, particles in section BC and particles in CD. Accordingly, upon receiving this message, P_1 will first update its interfacial particles in section A, then section AB then section AD.

Also, an extra agreement is agreed upon the order of the sections within each message that each section list is written in ascending order with reference to their particle ID.

Additionally, a safety feature is installed in the message. Each section of the message is further composed of two parts, the first item is an integer number n_0 describing the number of particles in the section. During updating, P_1 will first compare the n_o (of the number of particles in section C in P_0) in the received message and the n_0' in the local section A, if these $n_0 \neq n_0'$, then the program will exit with an error message.

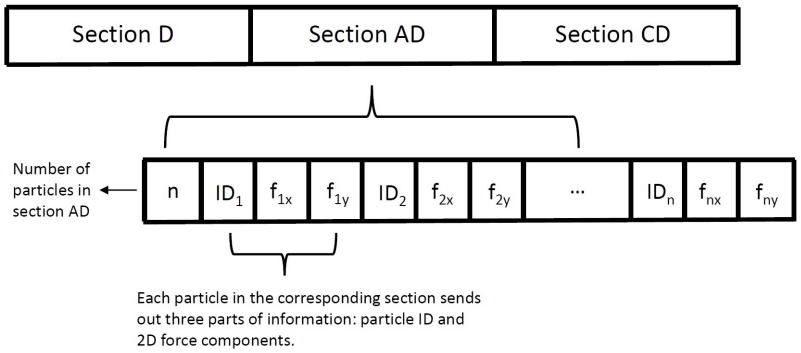

Figure 5.20: The vertical force transmission list is composed of three sub parts in which included particles in section D, AD and CD respectively.

Algorithm 5.2 Preparing a horizontal force transmission message.

```
1: integer nA, nAB, nAD                ▷ number of particles in each horizontal sections
2: integer size
3: size = 3 + 3 × (nA + nAB + nAD)
4: double sa[size]                     ▷ the send array of type double
5: integer i = 0
6: sa[i] = nA
7: i++
8: while parse ListA do
9:     sa[i] = (double) ID
10:    i++
11:    sa[i] = ax
12:    i++
13:    sa[i] = ay
14:    i++
15: end while
16: sa[i] = nAB
17: i++
18: while parse ListAB do
19:    load the particle ID, ax, ay as above
20: end while
21: sa[i] = nAD
22: i++
23: while parse ListAB do
24:    load the particle ID, ax, ay as above
25: end while
```

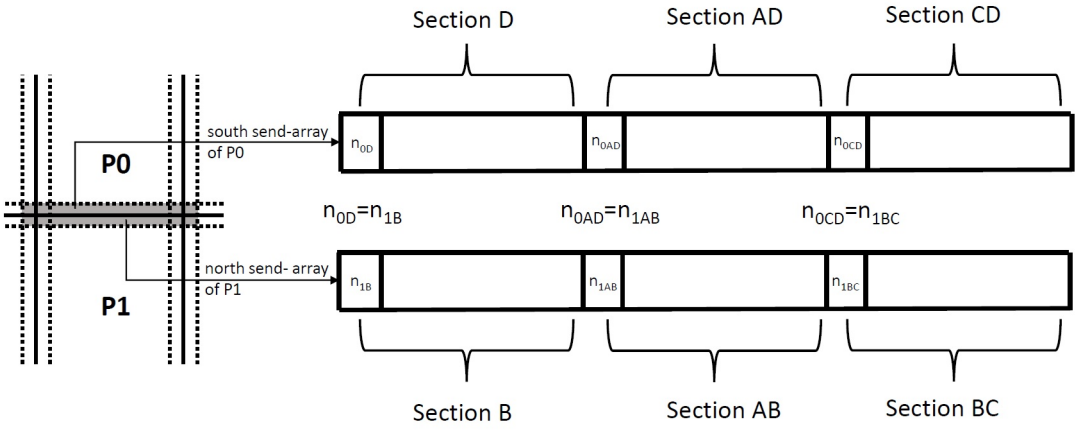

Figure 5.21: Between P_0 and P_1, all integer number (total number of particles in each section and particle ID) matches exactly.

5.6.3 Message Update

After communication in each direction, local particle velocity needs to be updated accordingly. During update, each local section list is called in according to the sequence the message is loaded. For example, if the received message is written in the sequence of ListB, then ListAB and lastly ListBC, local processor must also update ListB first, then ListAB and lastly ListBC accordingly. As explained in the message preparation section above, a safety issue (number of particles n_0 in each section and particle ID) is installed in the messages, if the received n_0 doesn't match the local n_0 or the received particle ID doesn't match the local one, then an error message is issued and program will exit. This feature is installed in spite of the fact that message passing is very expensive and therefore messages writing should by all means avoid redundancies, because as the simulation evolves over time, rounding error will eventually build up to a point that the same particle may behave different across processors, it is important to spot that point and halt the calculations.

5.7 Position Update

After force transmission, every particle in the domain should have the same value of net force across all the processors that hold it. Particle velocity is updated first, then particle position.

Algorithm 5.3 Velocity and position update.

1: **double** ax, ay ▷ 2D velocity acceleration component
2: **double** vx, vy ▷ 2D velocity component
3: **double** dx, dy ▷ 2D position component
4: **double** $deltat$ ▷ time step
5: $vx = vx + ax \times deltat$ ▷ update the velocity component for next time step
6: $vy = vy + ay \times deltat$ ▷ update the velocity component for next time step
7: $dx = dx + vx \times deltat$ ▷ update the position component for next time step
8: $dy = dy + vy \times deltat$ ▷ update the position component for next time step

5.8 Particle Migration

Migration is an operation that creates copies of a particle across processors by transmitting the vital information (e.g. The position and velocity) of a particle. This section

will provide a detailed explanation of how migration is implemented in the YNANO. The content of this section will be broken into two parts:

1. The judgement of migration.
2. Preparing, receiving and updating the MPI migration list.

The following section will be devoted to describe each part in detail.

5.8.1 The judgement of migration

After force transmission and position update, it is necessary for each processor to check its each sections (internal, A, B,C,D,AB,BC,CD and AD) the position of its local particles to see if any particles have left the its previous section and if any particles need to be migrated to adjacent processors. Migration is only necessary when new copies of a particle need to be created across adjacent processors.

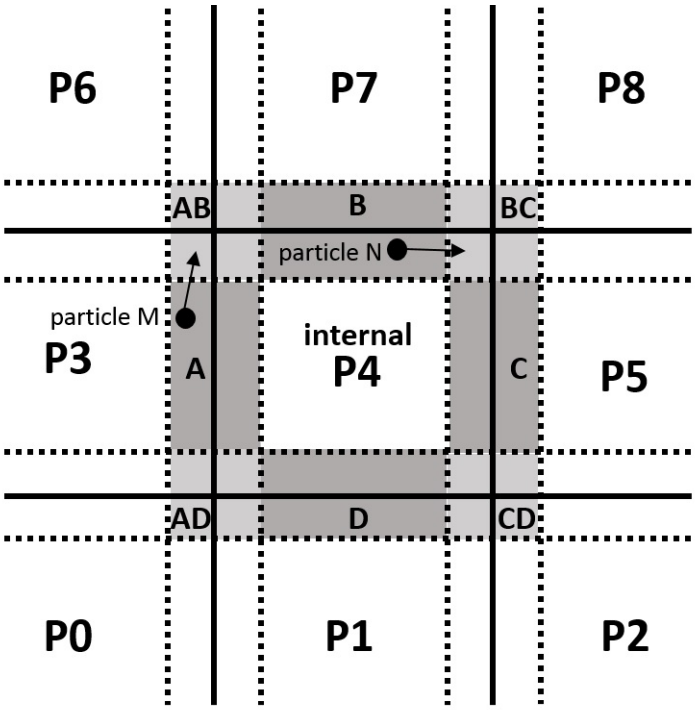

Figure 5.22: Particle *M* only needs to be migrated vertically. Particle *N* only needs to be migrated horizontally.

Algorithm 5.4 Composition of a horizontal message during force transmission

```
1: integer nA, nAB, nAD                    ▷ number of particles in each horizontal sections
2: double ra[size]                         ▷ the received array of type double
3: integer i = 0
4: if ra[i] = nA then
5:     i++
6: elseexit with error
7: end if
8: while parse ListB do
9:     if ra[i] = ID then
10:        i++
11:        ax = ax + ra[i]
12:        i++
13:        ay = ay + ra[i]
14:        i++
15:        move on to the next particle in the list
16:    elseexit with error
17:    end if
18: end while
19: if ra[i] = nAB then
20:    i++
21: elseexit with error
22: end if
23: while parse ListAB do
24:    if ra[i] = ID then
25:        i++
26:        ax = ax + ra[i]
27:        i++
28:        ay = ay + ra[i]
29:        i++
30:        move on to the next particle in the list
31:    elseexit with error
32:    end if
33: end while
34: if ra[i] = nBC then
35:    i++
36: elseexit with error
37: end if
38: while parse ListBC do
39:    if ra[i] = ID then
40:        i++
41:        ax = ax + ra[i]
42:        i++
43:        ay = ay + ra[i]
44:        i++
45:        move on to the next particle in the list
46:    elseexit with error
47:    end if
48: end while
```

The theoretical guidelines for migration can be summarized as follows:

- Previous (before position update) share-ship of particle $O\{P_i\}$; in which P_i is the rank of the processor.
- Current (after position update) share-ship of particle $C\{P_j\}$; in which P_j is the rank of the processor.

$C\{Pj\}$ is an empty set when and only when the particle in question becomes external, that is, when the particle moves out of the local domain. When this happens, the cleaning routine is called in to wipe clean the particle in question off the local processor. The cleaning routine includes three steps:

1. Clear object content.
2. Cut the empty object off from the Filled List.
3. Add the empty object into the Empty List.

For every item in $C\{P_j\}$ that doesn't belong to $O\{P_i\}$, new copy of the particle needs to be created in the P_j processor.

By comparing the position before and after the particle position update, theoretically speaking the following could happen:

1. A previously internal particle moves outside of the domain. This cannot happen according to the stability criteria.
2. A previously interfacial particle moves outside of the domain. In this case, no migration is necessary.
3. A previously interfacial particle moves into the internal region. No migration is needed in this case.
4. A previously shared-by-4 interfacial particles moves into a shared-by-2 region. No migration is required because in this case, copies of the particle in question need to be destroyed, and this doesn't require communication.
5. A previously shared-by-2 interfacial particle moves into a shared-by-4 region. Migration is required in this case either horizontally or vertically because new copies of the particle need to be created in the adjacent processors.
6. A previously internal particle moves into the interfacial region. This case is more complicated than meets the eye.

(a) Moving into shared-by-2 interfacial region. In this case, one new copy of the particle needs to be created in the neighbouring processor either horizontally or vertically.

(b) Moving into shared-by-4 interfacial region. In this case, 3 new copies of the particle need to be created in the neighbouring processors. Migration in both horizontal and vertical direction are required.

Algorithm 5.5 Decide which section the particle belongs to after position update.

1: **double** dx, dy ▷ 2D position component of a specific particle
2: **double** buf ▷ buffer size
3: **double** x_0, x_1, y_0, y_1 ▷ local boundaries
4: **double** $l_0, l_1, r_0, r_1, t_0, t_1, b_0, b_1$ ▷ local buffered boundaries for left, right, top and bottom
5: $l_0 = x_0 - buf$
6: $l_1 = x_0 + buf$
7: $r_0 = x_1 - buf$
8: $r_1 = x_1 + buf$
9: $t_0 = y_0 - buf$
10: $t_1 = y_0 + buf$
11: $b_0 = y_1 - buf$
12: $b_1 = y_1 + buf$
13: **if** $dx >= l_0$ and $dx <= l_1$ and $dy > b_1$ and $dy < t_0$ **then**
14: particle is in local Section A
15: **else if** $dx >= r_0$ and $dx <= r_1$ and $dy > b_1$ and $dy < t_0$ **then**
16: particle is in local Section C
17: **else if** $dx > r_1$ and $dx < l_0$ and $dy >= t_0$ and $dy <= t_1$ **then**
18: particle is in local Section B
19: **else if** $dx > r_0$ and $dx < l_0$ and $dy >= b_0$ and $dy <= b_1$ **then**
20: particle is in local Section D
21: **else if** $dx >= l_0$ and $dx <= l_1$ and $dy >= t_0$ and $dy <= t_1$ **then**
22: particle is in local Section AB
23: **else if** $dx >= r_0$ and $dx <= r_1$ and $dy >= t_0$ and $dy <= t_1$ **then**
24: particle is in local Section BC
25: **else if** $dx >= r_0$ and $dx <= r_1$ and $dy >= b_0$ and $dy <= b_1$ **then**
26: particle is in local Section CD
27: **else if** $dx >= l_0$ and $dx <= l_1$ and $dy >= b_0$ and $dy <= b_1$ **then**
28: particle is in local Section AD
29: **else if** $dx > l_1$ and $dx < r_0$ and $dy > b_1$ and $dy < t_0$ **then**
30: particle is in local Section I
31: **else if** ERROR MESSAGE **then**
32: **end if**

Algorithm 5.6 Possible new position for a particle previously in section A.

```
1: double dx, dy                                          ▷ a particle in Section A
2: while parse the local Section A list from last time step do
3:     if the particle moves to Section A then
4:         do nothing
5:     else if the particle moves to Section B then
6:         remove the current particle from ListA
7:         add it to ListB
8:         add it to B_v migration list.
9:     else if the particle moves to Section D then
10:        remove the current particle from ListA
11:        add it to ListD
12:        add it to D_v migration list.
13:    else if the particle moves to Internal then
14:        remove the current particle from ListA
15:        add it to ListI
16:    else if the particle moves to Section AB then
17:        remove the current particle from ListA
18:        add it to ListAB
19:        add it to AB_v migration list.
20:    else if the particle moves to Section AD then
21:        remove the current particle from ListA
22:        add it to ListAD
23:        add it to AD_v migration list.
24:    else if the particle moves out of the local boundary then
25:        remove the current particle from ListA
26:        add it to EmptyList
27:        clear its object content in the StackList.
28:    end if
29: end while
```

Algorithm 5.7 Possible new position for a particle previously in section I (Internal).

```
1:  double dx, dy                                          ▷ a particle in Section I
2:  while parse the local Section I list from last time step do
3:      if the particle moves to Section I then
4:          do nothing
5:      else if the particle moves to Section A then
6:          remove the current particle from ListI
7:          add it to ListA
8:          add it to A_h migration list.
9:      else if the particle moves to Section B then
10:         remove the current particle from ListI
11:         add it to ListB
12:         add it to B_v migration list.
13:     else if the particle moves to Section C then
14:         remove the current particle from ListI
15:         add it to ListC
16:         add it to C_h migration list.
17:     else if the particle moves to Section D then
18:         remove the current particle from ListI
19:         add it to ListD
20:         add it to D_v migration list.
21:     else if the particle moves to Section AB then
22:         remove the current particle from ListI
23:         add it to ListAB
24:         add it to AB_hv migration list.
25:     else if the particle moves to Section BC then
26:         remove the current particle from ListI
27:         add it to ListBC
28:         add it to BC_hv migration list.
29:     else if the particle moves to Section CD then
30:         remove the current particle from ListI
31:         add it to ListCD
32:         add it to CD_hv migration list.
33:     else if the particle moves to Section AD then
34:         remove the current particle from ListI
35:         add it to ListAD
36:         add it to AD_hv migration list.
37:     else if the particle moves out of the local boundary then
38:         remove the current particle from ListI
39:         add it to EmptyList
40:         clear its object content in the StackList.
41:     end if
42: end while
```

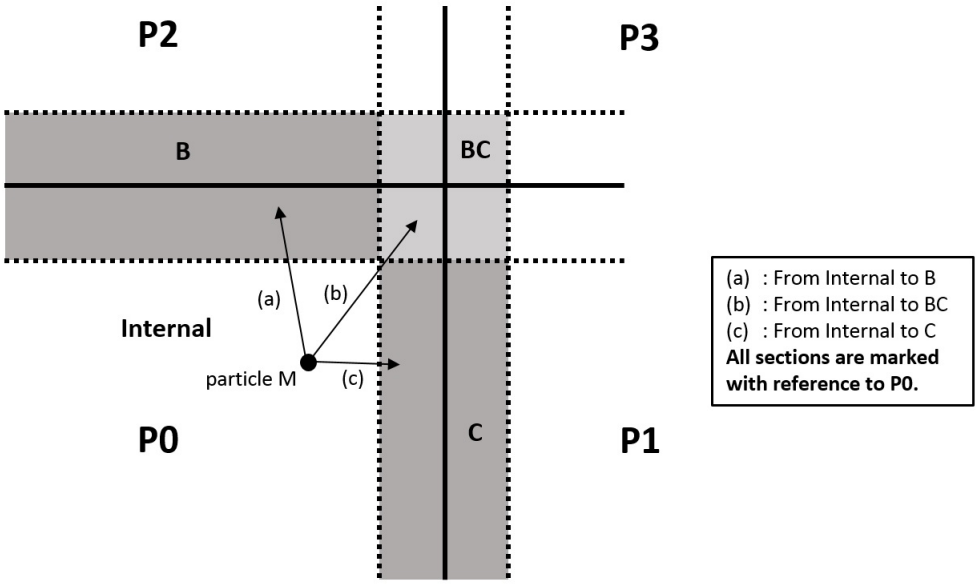

Figure 5.23: Particle M is an internal particle to P_0. It can move to either section B, C, or BC.

For a more concrete illustration, particle M in Figure 5.23 is

1. An internal particle in processor P_0 at beginning of the current time step, and it moves to

   (a) Section B: Shared-by-2 interfacial section after position update. What happens is that particle M is no longer an internal particle, it becomes an interfacial particle that is shared between P_0 and P_2. Share-ship of particle M is increased, a new copy of particle M needs to be created in P_2. In this case, particle M needs to be written in to the vertical migration list for B list.

   (b) Section BC: Shared-by-4 Corner region. It is clear that three new copied need to be created in the adjacent processors that share this corner. For this to happen, migration will take place in two directions subsequently, particle M will first be written into the horizontal message to be copied into P_1, then P_0 and P_1 will transmit particle M vertically to P_2 and to P_3 respectively. In the end, all four processor will have an identical copy of the interfacial particle M.

   (c) Outside of the domain, the program should exit with an error message, because according to the stability criteria, no single particle can move beyond 1 cell size in one time step.

2. A shared-by-2 particle M in section B of P_0 (and also shared by P_1) at beginning of the current time step, and it moves to:

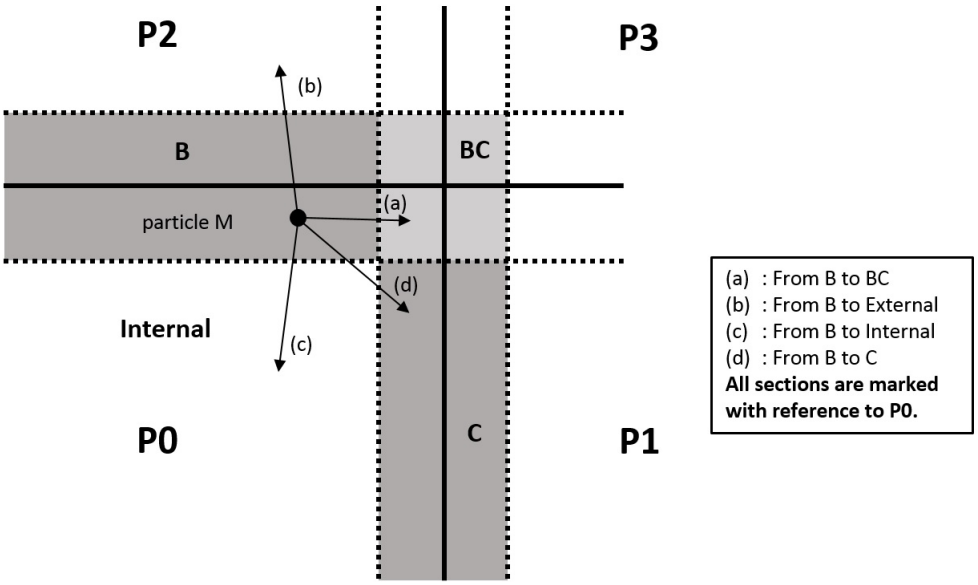

Figure 5.24: Particle M is in section B of P0. It can move to either section I, BC, C, or external.

(a) Section BC: Shared-by-4 corner region. In this case, particle M experiences an increase in share-ship, as it is now shared among all four processors, which means two more copies of particle M need to be created in P_1 and P_3, as P_0 and P_1 already have particle M locally. Therefore, P_0 and P_1 need to vertically transmit a copy of particle A to P_2 and to P_3 respectively.

(b) External. In this case, the share-ship of particle M is reduced therefore no migration is needed. Also, as processor P_0 will no longer retain a copy of particle M locally, the cleaning routine is called in.

(c) Section B: shared-by-2 region. This case involves a change of share-ship. Previously, particle M is shared between P_0 and P_1, now it is to be shared between P_0 and P_2. Therefore, particle M needs to be vertically migrated into P_2.

3. A shared-by-4 particle M in section BC in P_0 (and also shared all adjacent processors) at beginning of the current time step, and it moves to

(a) Section B: Shared-by-2 region. In this case, particle M experiences a reduction in share-ship, and therefore no migration is needed.

(b) Internal. Particle M clearly undergoes a reduction in its share-ship in this case, therefore no migration is needed.

(c) External. Cleaning routine is called in. Same as 2.b) above.

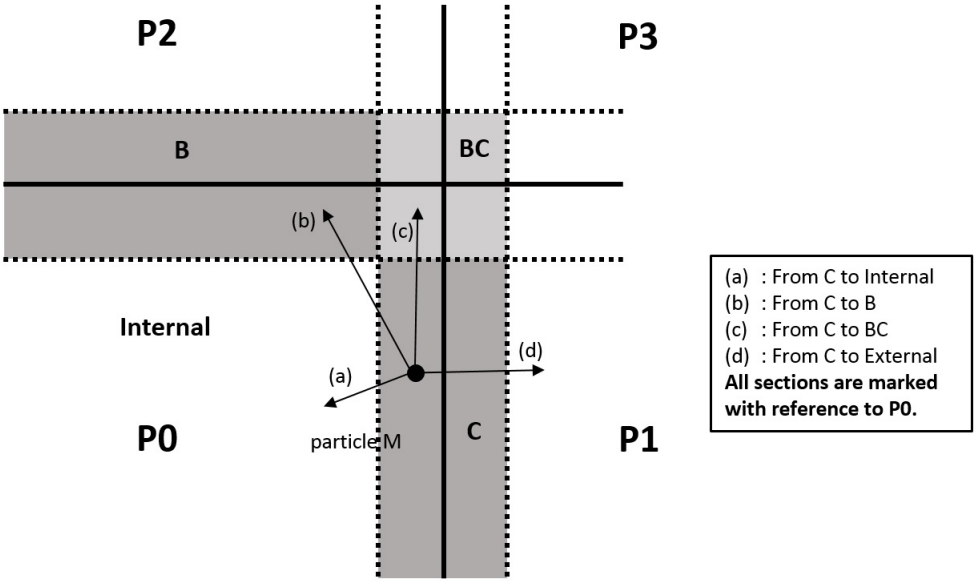

Figure 5.25: Particle M is in section C of P_0. It can move to either section I, BC, B, or external.

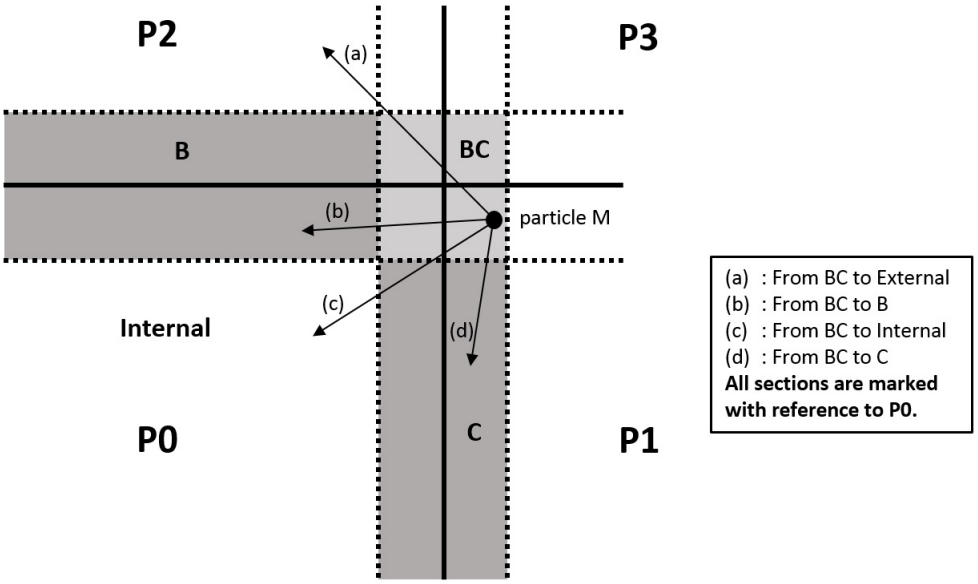

Figure 5.26: Particle M is in section BC of P_0. It can move to either section I, B, C, or external.

5.8.2 Preparing, receiving and updating the migration list

The message needs to include vital information as atomic ID, 2*D* position and 2*D* velocity information of a particle.

Horizontal migration differs from the vertical migration in that some particles might only need to be migrated horizontally, and some particles might need to be migrated both horizontally and vertically. Therefore the design of the migration list should reflect that distinction.

For instance, the horizontal east-wards migration list for local processor is consisted of 5 categories, particles in section C, particle in section BC that only migrates horizontally, particles in BC that need to migrate horizontally and as well as vertically, particles in CD that only migrates horizontally, and lastly, particles in section CD that need to migrate horizontally and as well as vertically.

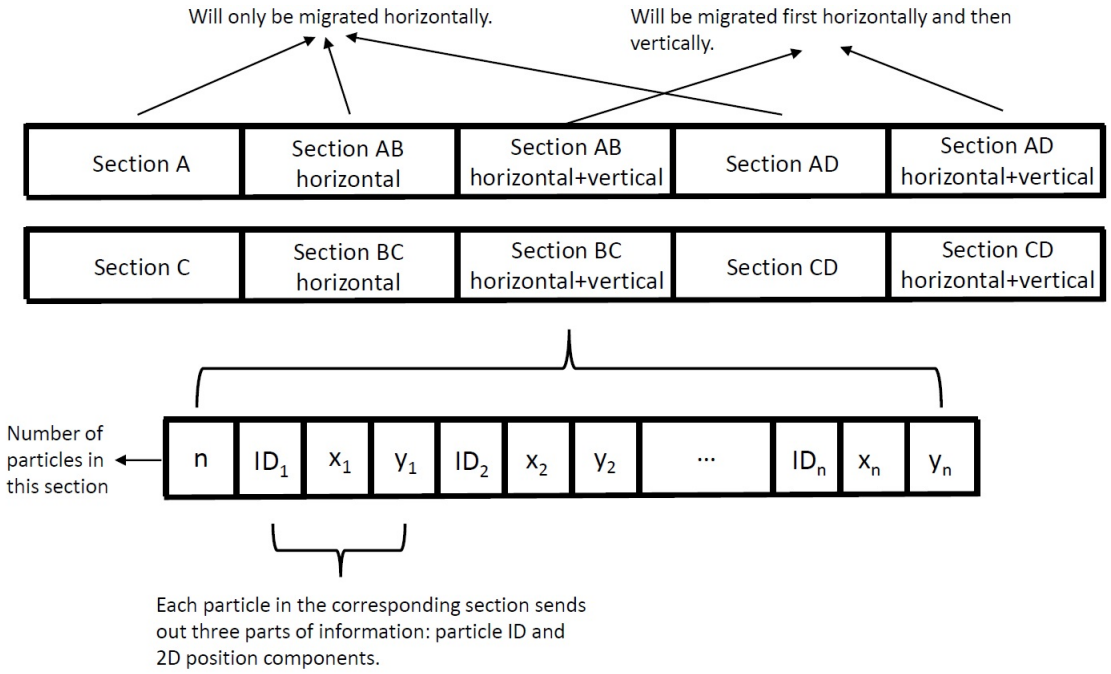

Figure 5.27: A horizontal message of migration is composed of five parts.

In a close-up look at Figure 5.27, each category consists of two parts, the first is an integer number indicating how many particles are there in this category, the second part is an array of integer number of particle ID, and double numbers of 2*D* position and 2*D* velocity.

Algorithm 5.8 Composition of a horizontal migration message.

1: **integer** nA, $nABh$, $nABhv$, $nADh$, $nADhv$ ▷ number of particles in each send mode of sections
2: **integer** $size$
3: $size = 5 \times (nA + nABh + nABhv + nADh + nADhv)$
4: **double** $sa[size]$ ▷ the send array of type double
5: **integer** $i = 0$
6: $sa[i] = nA$
7: $i++$
8: **while** parse the Section A particle list **do** ▷ particles that only migrate horizontally
9: $sa[i] = (double)\, ID$
10: $i++$
11: $sa[i] = dx$
12: $i++$
13: $sa[i] = dy$
14: $i++$
15: $sa[i] = vx$
16: $i++$
17: $sa[i] = vx$
18: $i++$
19: **end while**
20: $sa[i] = nABh$
21: $i++$
22: **while** parse the AB_h particle list **do** ▷ particles that only migrates horizontally
23: load the particle ID, dx, dy, vx, vy as above
24: **end while**
25: $sa[i] = nABhv$
26: $i++$
27: **while** parse the AB_hv particle list **do** ▷ particles that migrate horizontally and vertically
28: load the particle ID, dx, dy, vx, vy as above
29: **end while**
30: $sa[i] = nADh$
31: $i++$
32: **while** parse the AD_h particle list **do** ▷ pariticles that only migrate horizontally
33: load the particle ID, dx, dy, vx, vy as above
34: **end while**
35: $sa[i] = nADhv$
36: $i++$
37: **while** parse the AD_hv particle list **do** ▷ particles migrate horizontally and vertically
38: load the particle ID, dx, dy, vx, vy as above
39: **end while**

Upon receiving the horizontal migration, two operations need to be done.

1. Parse the particle contents in the horizontal migration list, and add new immigrant particles into the local NewP List.
2. Add the immigrant particles that need to migrate vertically into the vertical migration list.

To add the particle into local memory, three steps need to be taken:

1. Object class will be freed (by cutting off linkages) from the Empty List,
2. This newly freed object will be added into the NewP List and as well as the section list (e.g. section A, B, C, D, and corner),
3. Atomic ID, 2D position and 2D velocity will be written into the object.

The vertical migration list is simpler than the horizontal list because there's no further migration operations after it. For instance, a typical vertical north-wards migration list is shown as follows, it consists of 3 categories: particles in section B, particles in section AB, particles in section BC. And by the same principle, each category has 2 parts with a heralding integer number indicating the total number of particles in that category.

Updating the vertical list is the same as that of the horizontal list.

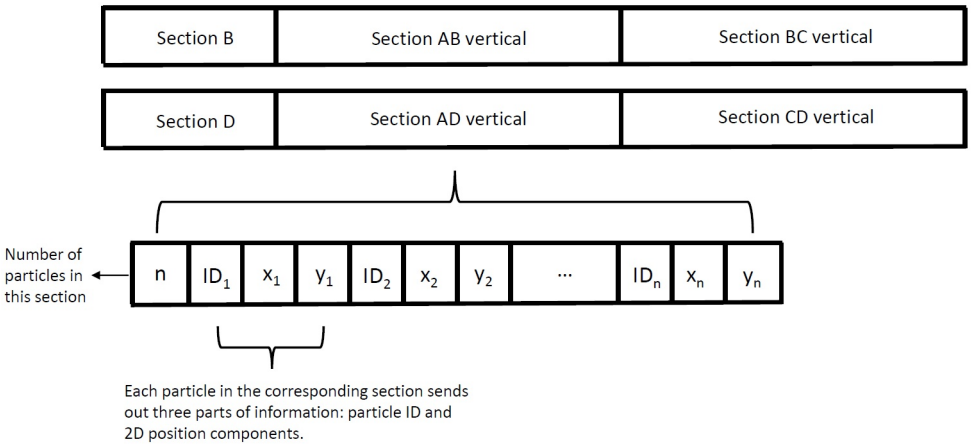

Figure 5.28: A vertical message of migration is composed of three parts.

Algorithm 5.9 Update the received horizontal migration message.

1: **integer** nA, $nABh$, $nABhv$, $nADh$, $nADhv$ ▷ number of particles in each send mode of sections
2: **integer** $size$
3: $size = 5 \times (nB + nABv + nCDv)$
4: **double** $sa[size]$ ▷ the send array of type double
5: **integer** $i = 0$
6: $sa[i] = nB$
7: $i++$
8: **while** parse the Section B particle list **do**
9: $sa[i] = (double)\,ID$
10: $i++$
11: $sa[i] = dx$
12: $i++$
13: $sa[i] = dy$
14: $i++$
15: $sa[i] = vx$
16: $i++$
17: $sa[i] = vx$
18: $i++$
19: **end while**
20: $sa[i] = nABv$
21: $i++$
22: **while** parse the AB_v particle list **do**
23: load the particle ID, dx, dy, vx, vy as above
24: **end while**
25: $sa[i] = nBCv$
26: $i++$
27: **while** parse the AB_v particle list **do**
28: load the particle ID, dx, dy, vx, vy as above
29: **end while**

5.9 Communications

Communication is the most important and the most time-consuming part in a distributed-memory based parallelization solution, in this sense, the communication design of a code greatly affects its working efficiency. There are three key elements in communication: message preparation, message passing pattern, and message update.

Message should be designed in a concise way, messages should only contain essential information to reduce overhead cost. However as accurateness overrules efficiency,

safety-alarm issues should also be considered in the message design in that the program will halt and exit with an error message should there be an error.

Message passing pattern is important due to the intrinsic limitation of the way processors work, processors could only do one thing at a time, it either receives or sends. To achieve high performance efficiency, careful design on the pattern processors talk to each should be made.

Message update is intrinsically a result from message preparation and message passing pattern, and this in term influences how messages should be prepared.

The following section is devoted entirely to describe the first and second issue.

5.9.1 Message passing scheme

The objective of message passing is to make sure any changes to a common variable is reflected in all processors that (should) hold it (in force transmission and particle migration). A two-step communication scheme is deployed: First, horizontal communication. At the end of this step, all horizontally neighbouring processors will have the same value for the same shared variable. Second, vertical communication. At the end of this step, all vertically (and horizontally) neighbouring processors will have the same value for the same shared variable.

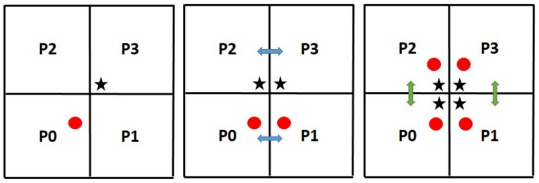

Figure 5.29: The two step communication scheme for particle migration. The circle in P_0 and the star in the P_3 are to be shared by all 4 neighbouring processors in the next time step. During horizontal migration, P_0 sends P_1 the circle, and P_3 sends P_3 the star. During vertical migration, P_0 and P_1 transmit the circle to P_2 and P_3, P_2 and P_3 transmit the star to P_0 and P_1. In the end, all four processors have a star and a circle.

One processor can communicate with only one other processor at any given time, when P_i and P_{i+1} exchange messages among themselves, neither of them could communicate with any other processors. To maximize the MPI efficiency, it is important to design the communicating pairs properly to make sure as many processors are mobilized as possible. In this thesis, the following message passing scheme is adopted.

First, each processor will generate two sub ranks r_x and r_y as follows:

$$r_x = (Pi)\%icol \tag{5.1}$$

$$r_y = (Pi)/icol \tag{5.2}$$

Horizontal message passing involves two stages, each stage includes two steps. Firstly, all processors Pi with an even r_x (except when $(r_x + 1) = icol$) will open up a channel to communicate (send and receive message) with its right hand side neighbour P_{i+1}, secondly, all processors P_i with an odd r_x will open up a channel to communicate (send and receive message) with its left hand side neighbour P_{i-1}. After the aforementioned two steps, one set of send and receive routine is completed between processor P_i with an even r_x and its right hand side neighbour. Now, communication needs to be carried out between processor P_i with an even r_x and its left hand side neighbour. By the same principle, all processors P_i with an even r_x (except when r_x) will open up a channel to communicate (send and receive message) with its left hand side neighbour P_{i-1}, secondly, all processors P_i with an odd r_x (except when $r_x = icol$) will open up a channel to communicate (send and receive message) with its right hand side neighbour P_{i+1}.

by the same principle, vertical message passing also involves two stages, each stage includes two steps. First, all processors P_i with an even r_y (except when (r_y) will open up a channel to communicate (send and receive message) with its neighbour P_{i+icol} immediately above it, secondly, all processors P_i with an odd r_y will open up a channel to communicate (send and receive message) with its neighbour P_{i-icol} immediately beneath it. After the aforementioned two steps, one set of MPI send and receive routine is completed between processor P_i with an even r_y and its neighbour immediately above it. Now, communication needs to be carried out between processor P_i with an even r_y and its neighbour immediately beneath it. By the same principle, all processors P_i with an even r_y (except when r_y) will open up a channel to communicate (send and receive message) with its neighbour P_{i-icol} immediately beneath it, secondly, all processors P_i with an odd r_y (except when $r_y = irow$) will open up a channel to communicate (send and receive message) with its neighbour P_{i+icol} immediately above it.

Algorithm 5.10 Horizontal message passing in four steps using MPI_SendRecv.

```
1: integer rx, ry                                              ▷ an integer number
2: integer MyRank                                              ▷ a double number
3: integer icol, irow                                          ▷ a double number
4: if (rx%2 == 0) and (rx! = (icol − 1)) then
5:     Message passing with processor with a rank MyRank + 1
6: end if
7: if (rx%2 == 1) then
8:     Message passing with processor with a rank MyRank − 1
9: end if
10: if (rx%2 == 0) and (rx! = 0) then
11:    Message passing with processor with a rank MyRank − 1
12: end if
13: if (rx%2 == 1) and (rx! = (icol − 1)) then
14:    Message passing with processor with a rank MyRank + 1
15: end if
```

Algorithm 5.11 Vertical message passing in four steps using MPI_SendRecv.

```
1: integer rx, ry                                              ▷ an integer number
2: integer MyRank                                              ▷ a double number
3: integer icol, irow                                          ▷ a double number
4: if (ry%2 == 0) and (ry! = (irow − 1)) then
5:     Message passing with processor with a rank MyRank + icol
6: end if
7: if (ry%2 == 1) then
8:     Message passing with processor with a rank MyRank − icol
9: end if
10: if (ry%2 == 0) and (ry! = 0) then
11:    Message passing with processor with a rank MyRank − icol
12: end if
13: if (ry%2 == 1) and (ry! = (irow − 1)) then
14:    Message passing with processor with a rank MyRank + icol
15: end if
```

5.9.2 Message passing technique

As mentioned above, opening up a channel is expensive in MPI, therefore it pays to utilize the opened channel as much as possible. To send a message using MPI_Send opens up a channel, and to receive a message using MPI_Recv opens up the channel again. In this thesis, MPI_SendRecv is used as it both sends and receives a message during the time the channel is open.

5.9.3 Message preparation

Processors communicate with each other by sending messages in an array. When all the information are of the same type, for example, integer or double, it is easy to initialize the sending array straightaway as type integer or type double. However, if the information is a combination of various types, derived data type needs to be created to incorporate different types of variables. However, derived data type could be expensive, to deal with this, a simple solution is developed in this thesis.

In this thesis, during all communications, two types of variables are transmitted across all processors, type integer and type double. For instance, a force transmission message is composed of three categories of information, total number of particles in each section, particle ID, and particle force, the first two are of type integer, the last is of type double. Likewise during migration, three categories of information are present in every message, particle ID, particle velocity and particle position, the first is of type integer, the latter two are of type double. A simple solution to send messages without constructing a new derived type is by casting the integer into double during message writing, and casting it back after receiving it.

Every integer number is first cast into double, and subsequently added by 0.1, the reason why the addition is necessary is to avoid the inevitable random rounding errors in message passing. Upon receiving the message, all these transformed double data are casted back into integer.

Algorithm 5.12 Casting the particle ID of type integer into type double.

1: **integer** idi ▷ an integer number
2: **double** idd ▷ a double number
3: $idd = ((double)\, idi) + 0.1$

Algorithm 5.13 Casting the received particle ID of type double back into type integer.

1: **integer** idi ▷ an integer number
2: **double** idd ▷ a double number
3: $idi = (int)\, idd$

5.10 Parallelization of Input

There are two strategies to initialize the particle position and velocity in the sub-domain of each local processors.

- Each processor reads in the same input file instructions and generate its own particles according to its ranks.
- One root processor generates all particles, it then divides and writes them into individual input files to be read by individual processors. Each processor will then read in the input file the root processor writes.

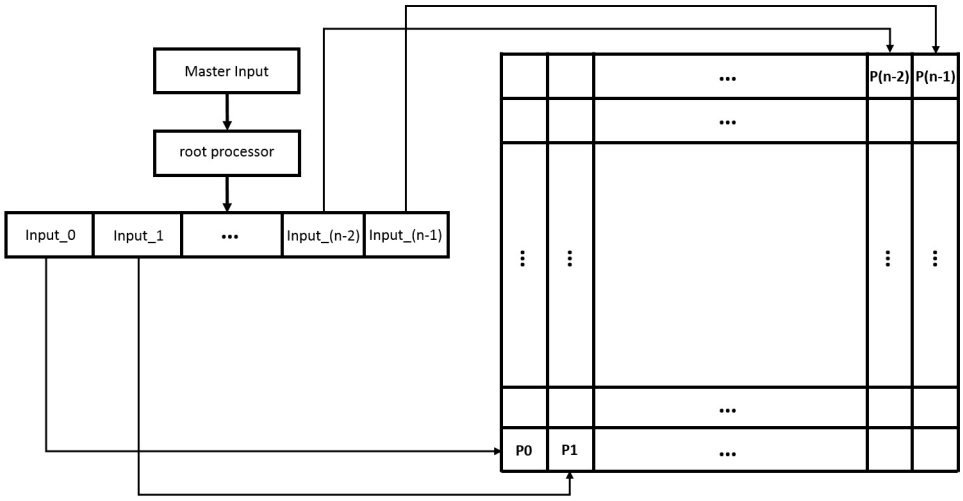

Figure 5.30: Parallel input flowchart. The root processor reads in master input file and generates all sub input files for each processors. Each processor then reads in its individual input file for particle initialization and system parameter setup.

In this thesis, the second approach is used. All particles are first generated by the root processor (P_0), P_0 then parses the whole particle list to decide which processor(s) this particle belongs to according to its geometric positions. There are four possible outcomes: 1) a particle has only one copy across all processors; 2) a particle is shared by two horizontally adjacent neighbouring processors; 3) a particle is shared by two vertically adjacent neighbouring processors; 4) a particle is shared by four adjacent neighbouring processors that revolve around one corner.

Algorithm 5.14 Deriving the *rx* and *ry* for local processor.

```
1: integer sx, sy                          ▷ The 2D velocity acceleration components of a particle
2: integer rankx, ranky                    ▷ The 2D velocity components of a particle
3: integer icol, irow
4: double delx, dely
5: double dx, dy
6: double wx, wy                           ▷ The 2D velocity components of a particle
7: double xmin, xmax, ymin, ymax           ▷ The 2D velocity components of a particle
8: double buf
9: integer flag
10: delx = (xmax − xmin)/icol
11: dely = (ymax − ymin)/irow
12: integer i, j
13: sx = 10
14: sy = 10
15: while parsing the particle list do
16:     for i = 0; i < icol − 1; i++ do
17:         wx = xmin + (i + 1) × deltax
18:         if dx >= (wx − buf) and dx <= (wx + buf) then
19:             sx = 1
20:             break
21:         else if dx < (wx − buf) then
22:             sx = 0
23:             break
24:         end if
25:     end for
26:     if sx == 10 then
27:         i = icol − 1
28:         sx = 0
29:     end if
30:     rankx = i
31:     for j = 0; j < irow − 1; j++ do
32:         wy = ymin + (j + 1) × dely
33:         if dy >= (wy − buf) and dy <= (wy + buf) then
34:             sy = 1
35:             break
36:         else if dy < (wy − buf) then
37:             sy = 0
38:             break
39:         end if
40:     end for
41:     if sy == 10 then
42:         j = irow − 1
43:         sy = 0
44:     end if
45:     ranky = j
46: end while
```

Algorithm 5.15 Writing individual input file according to particle's geometric position.

```
1: integer sx, sy                          ▷ The 2D velocity acceleration components of a particle
2: integer rankx, ranky                    ▷ The 2D velocity components of a particle
3: integer icol, irow                      ▷ The 2D velocity components of a particle
4: integer flag
5: while parse the particle list do
6:     if sx == 1 and sy == 1 then         ▷ section shared by four processors
7:         integer localp[4]
8:         localp[0] = ranky × icol + rankx
9:         localp[1] = ranky × icol + rankx + 1
10:        localp[2] = (ranky + 1) × icol + (rankx + 1)
11:        localp[3] = (ranky + 1) × icol + rankx
12:        flag = 4
13:        for int fi = 0; fi < 4; fi++ do
14:            write into individual input file end with localp[fi]
15:        end for
16:    else if sx == 1 and sy == 0 then    ▷ section shared by 2 horizontally adjacent processors
17:        integer localp[2]
18:        localp[0] = ranky × icol + rankx
19:        localp[1] = ranky × icol + rankx + 1
20:        flag = 3
21:        for int fi = 0; fi < 2; fi++ do
22:            write into individual input file end with localp[fi]
23:        end for
24:    else if sx == 0 and sy == 1 then    ▷ section shared by two vertically adjacent processors
25:        integer localp[2]
26:        localp[0] = ranky × icol + rankx
27:        localp[1] = (rank + 1) × icol + rankx
28:        flag = 2
29:        for int fi = 0; fi < 2; fi++ do
30:            write into individual input file end with localp[fi]
31:        end for
32:    else if sx == 0 and sy == 0 then    ▷ section shared by only one processor
33:        integer localp[0]
34:        localp[0] = ranky × icol + rankx
35:        flag = 1
36:        for int fi = 0; fi < 1; fi++ do
37:            write into individual input file end with localp[fi]
38:        end for
39:    end if
40: end while
```

5.11 Parallelization of Output

Since interfacial particles are shared among neighbouring processors, it is important to make sure that no particle is written into output file more than once. In order to do so, each processor needs to agree upon the following output rules:

- Write internal particles (particles in section A).
- Write interfacial particles situated in the north shared-by-2 border (section B).
- Write interfacial particles situated in the east shared-by-2 border (section C).
- Write interfacial particles situated in the north-east shared-by-4 corner (section BC).

As explained in the domain decomposition section, each local particle belongs to a section list according to their position, therefore output is done by parsing and writing out the particles in section list Internal, B, C and BC.

As can be seen from Figure 5.31, for a 9 processor network, P_8 will only write output for its internal particles, P_0 will write out its internal particles and interfacial particles in section B, C and BC.

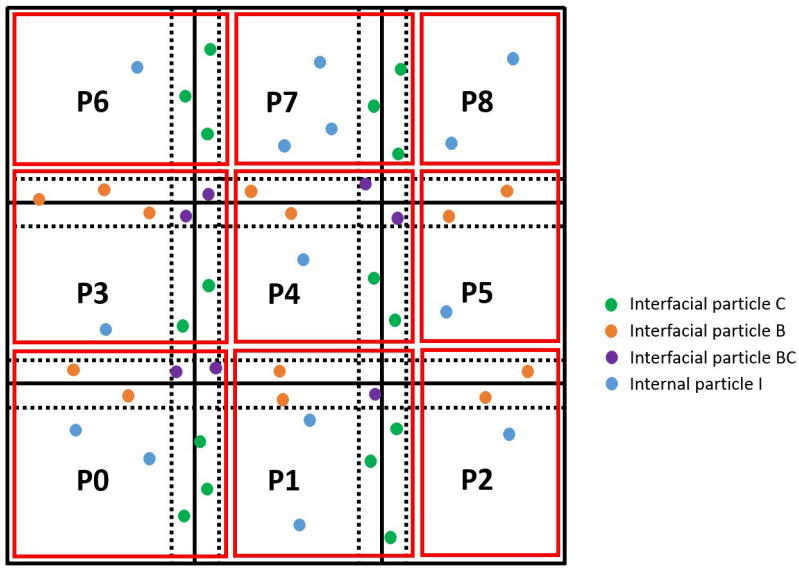

Figure 5.31: Output for each processor: internal particle I, interfacial particles B in the north border, interfacial particles C in the east border and interfacial particles BC in the north-east corner.

Chapter 6

VALIDATION AND PERFORMANCE TESTS OF THE DEVELOPED YNANO PARALLEL SOLUTIONS

6.1 Introduction

The main objective of parallelization is to improve performance, it is necessary to test the developed parallelization solutions of YNANO and assess its performance. For verification purposes, a study of the system kinetic energy evolution is presented, comparisons of the results are also made across systems running on different number of processors. Further verification and performance tests are presented in Chapter7.

All examples in this thesis were run on an HPC cluster with 3592 nodes. Each node contains two 8-core 2.6 GHz Intel Xeon $E5-2670$ CPUs. 2395 out of the 3592 nodes have 32 GB of RAM each, 1125 out of the rest nodes have 64 GB of RAM each and the rest 72 nodes have 128 GB of memory each. In other words, the maximum RAM available for each processor is 8 GB.

6.2 Numerical Example

A $2D$ box filled with $250,000$ argon atoms is tested. The global initial temperature is set at $90K$. The initial state of the argon atoms is of gaseous state (rarefied gas). The example is tested on up to 64 processors. Recorded simulation time for different numbers of processors and calculated speedup are presented in Table 6.1. Simulation times and speedup for up to 64 processors are plotted in Figure 6.2 and Figure 6.3.

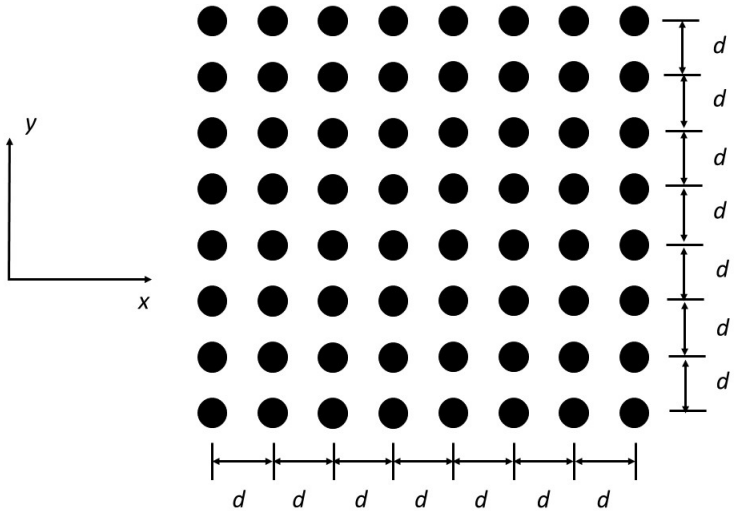

Figure 6.1: $250,000$ rarefied argon gas atoms are boxed in a $9120\mathring{A}$ by $9120\mathring{A}$ container. The initial inter-particle spacing is $36\mathring{A}$, the initial system temperature is set at $90K$. The system is let to rest into equilibrium on 1, 4, 8, 16, 32 and 64 cores.

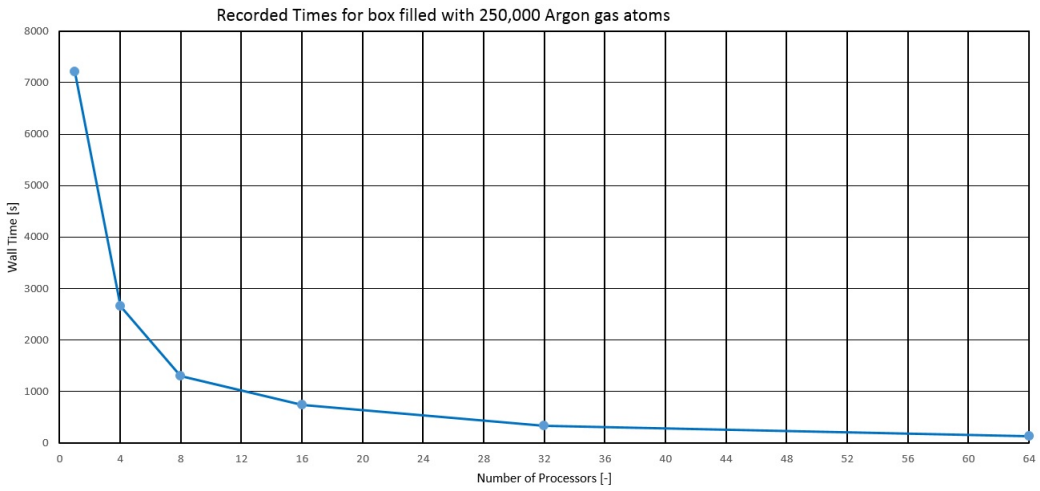

Figure 6.2: Recorded CPU time for a box filled with $250,000$ Argon atoms.

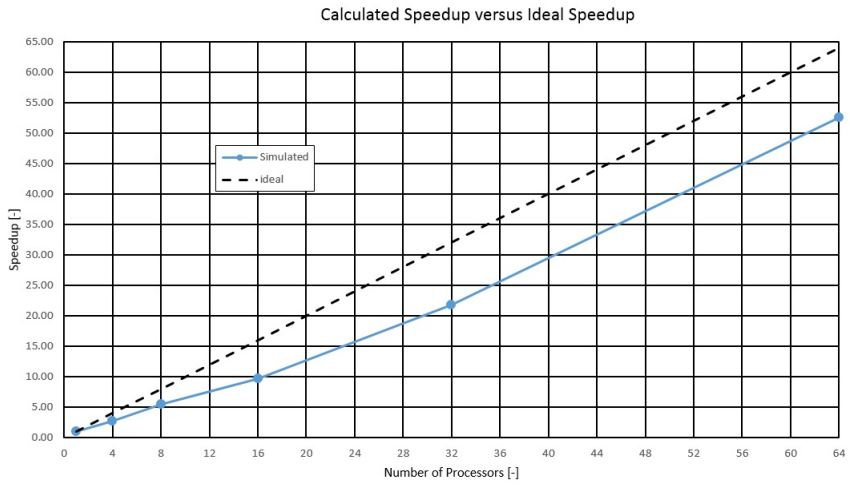

Figure 6.3: Calculated speedup for a box filled with 250,000 Argon atoms.

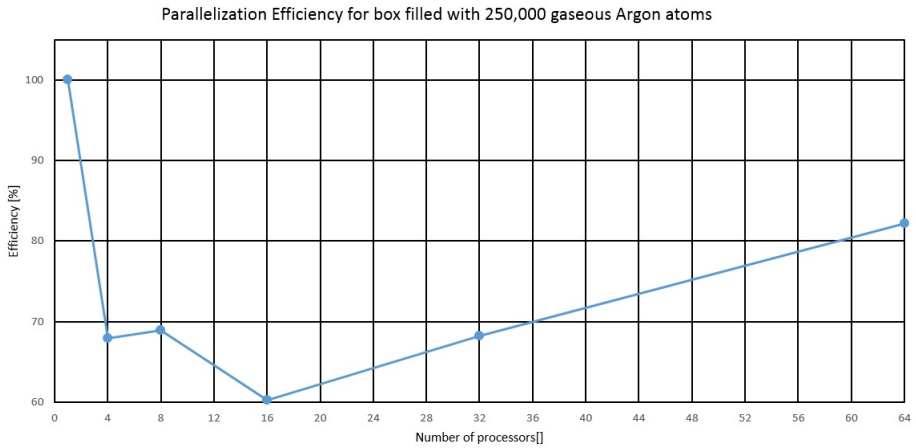

Figure 6.4: Calculated efficiency for a box filled with 250,000 Argon atoms.

Number of Processors	Wall Time [s]	Speedup [-]	Efficiency [%]
1	7205	1	100
4	2651	1.71	67.95
8	1307	4.51	68.91
16	748	8.63	60.20
32	330	20.83	68.23
64	127	56.73	88.64

Table 6.1: Recorded CPU times and calculated speedup for a box filled with 250,000 Argon atoms.

The recorded time and efficiency for different number of processors and the calculated speedup are summarized in Table 6.1. Simulation time, efficiency and calculated

speedup up to 64 cores are plotted in Figure 6.2, Figure 6.4 and Figure 6.3.

This test case could be considered as a worst case scenario in communication overhead cost, because first of all the computational cost (particle interaction) for rarefied gas interaction is very few, and secondly since rarefied gas is free to move randomly around the box, the communication between neighbouring processors for particle migration is expected to be huge.

For simulations with 4 cores, the entire domain is divided equally into 4 subdomains with 2 rows and 2 columns. For simulations with 8 cores, the domain is broken up into 8 equal sized subdomains with 4 rows and 2 columns (or 2 rows and 4 columns). Similarly, 16-core domain has 4 columns and 4 rows, 32-core domain has 4 rows and 8 columns (or 8 columns and 4 rows), and finally, 64-core domain has 8 columns and 8 rows.

The speedup for 4 cores and 8 cores are very small, because the surface of the subdomain is quite big, therefore the size of message to be exchanged between neighbouring cells is huge, thus creating the communicational overhead cost. The speedup between 8 processors and 64 processors has a close to linear trend.

The efficiency of the test case keeps increasing as the number of processors increases, especially for cases with 32 and 64 processors, the resultant efficiency are surprisingly high, even though they are expected to fall as the ratio between local work load and communicational message size diminishes as the number of processors increases, which suggests that the efficiency saved from work sharing among processors is overtaken by the overhead cost of communications between processors. The resulted efficiency should most probably result from the implementation details of the HPC cluster.

Due to the huge size of the simulation dimension, a picture trying to depict the whole domain will present no meaningful information, and since the simulation is horizontally and vertically homogeneous, only a quarter of the simulation domain is presented for dynamic evolution study and for comparison study across different number of processors. A motion sequence for the lower left quarter of a box filled with $250,000$ executed on 64 processors at 16 different times is shown in Figure 6.5. The 64 processors are distributed evenly into 8 rows and 8 columns.

It should be noted that in the above example, the buffer zone size specified in the previous chapters is the size of the cutoff radius of Lennard Jones $12-6$ potential for Argon atom.

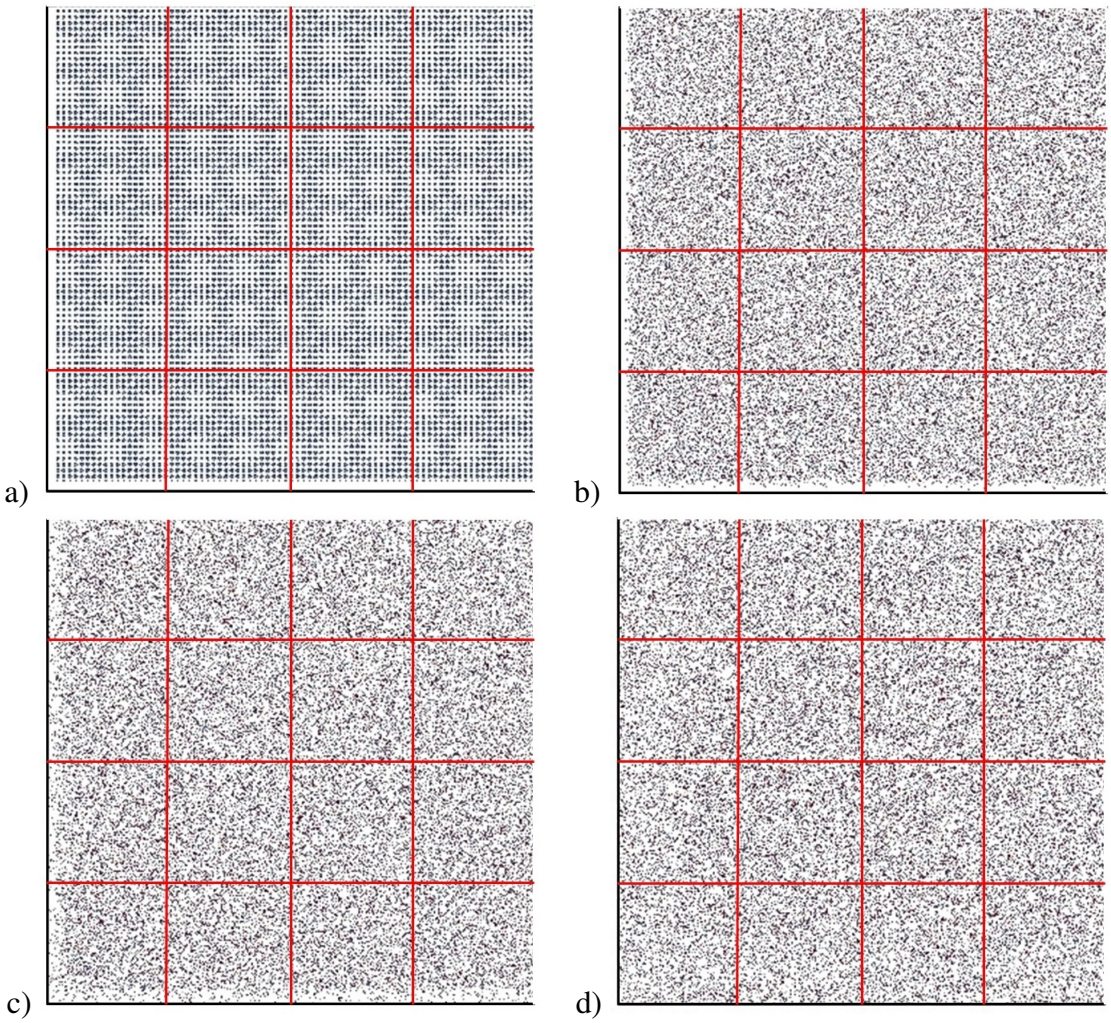

Figure 6.5: A motion sequence for the lower left quarter of a box filled with 250,000 Gaseous Argon particles executed on 16 processors. a) Time 0 *ps*, b) Time 10 *ps*, c) Time 25 *ps*, d) Time 99 *ps*.

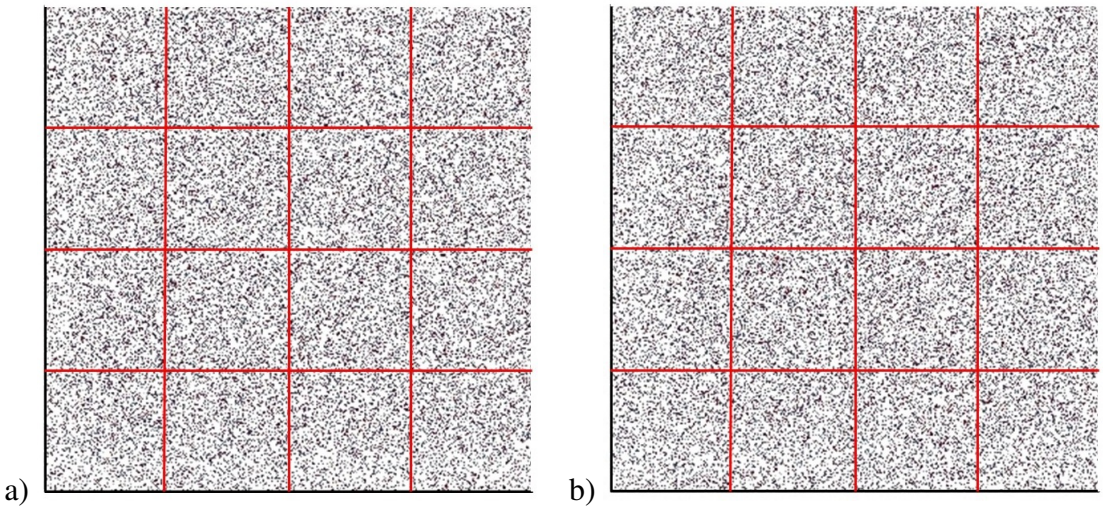

Figure 6.6: A motion sequence for a box filled with 250,000 Gaseous Argon particles executed on 4 processors. a) Time 80 *ps*, b) Time 99 *ps*.

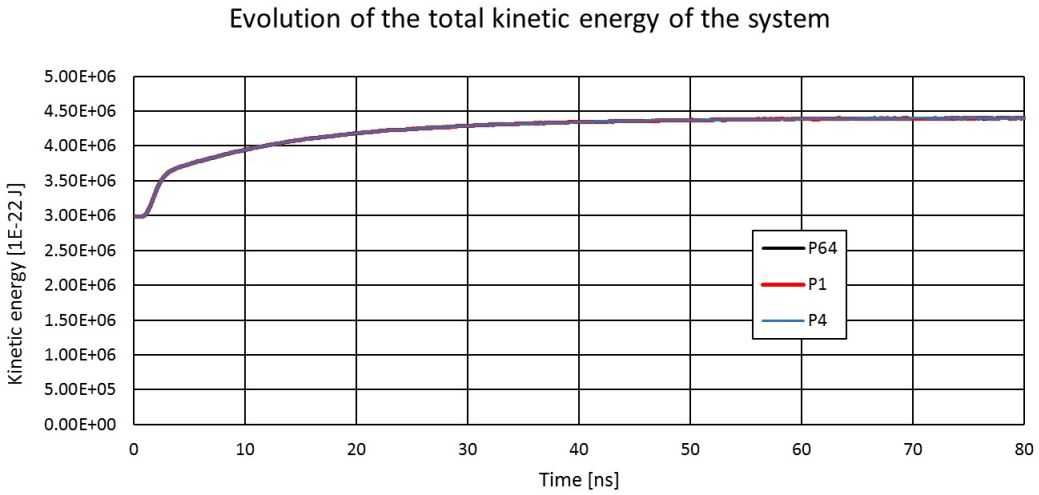

Figure 6.7: Total kinetic energy of the system of a box filled with 250,000 Argon atoms for the whole simulation time.

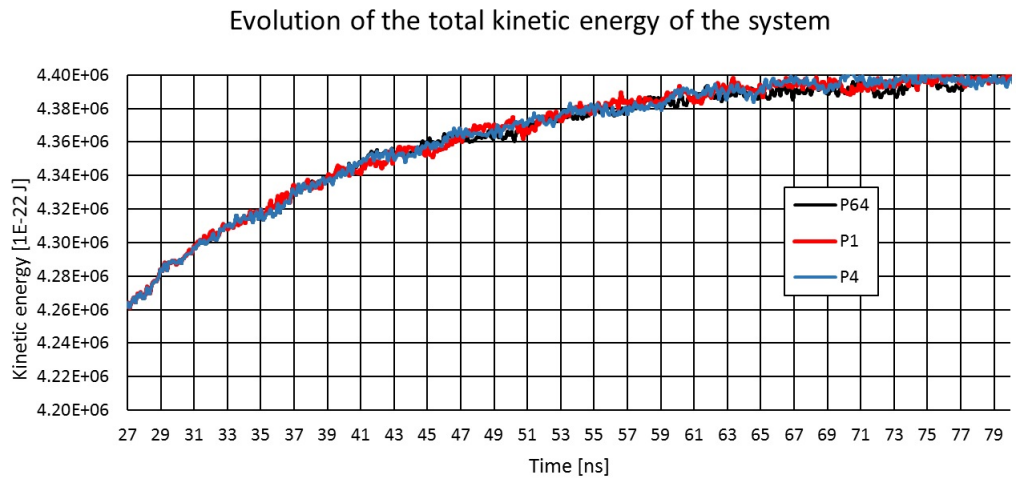

Figure 6.8: Total kinetic energy of the system of a box filled with 250,000 Argon atoms from 27 ns to 80 ns.

The test case is in effect simulating the gaseous diffusion process. Given time, the Argon gas particles are expected to permeate and blend homogeneously inside the box. The evolution of kinetic energy obtained from different numbers of processors are compared with that obtained from a sequential code, and the results show a good agreement. It could be seen from the evolution of the system kinetic energy that at $time = 80ps$, the system has reached equilibrium, and from the evolution of particle motion, the system also appears to be homogeneous at $time = 80ps$. The difference in the system kinetic energy is evidently a result of rounding errors.

The general trend of the evolution of the system is preserved in all simulations. The onset of discrepancy of kinetic energy happens at $29.3505\,ps$, but after the system reaches equilibrium, the discrepancy is quite small considering the highly volatile nature of the test case. This shows that the proposed parallelization solution converge well with the sequential version, this also suggests that the proposed parallelization solution is an accurate alternative to the sequential code and could be used in future simulations with significantly less CPU time required.

6.3 Conclusion

The performance of the testing example shows good scalability for the parallelized YNANO considering the fact that the testing example is a worst scenario dynamic case. Further test are carried out in Chapter 7.

The comparison of the results (particle position and system kinetic energy) of the

system also shows good agreement, thus confirming the validity of the developed parallelization solutions.

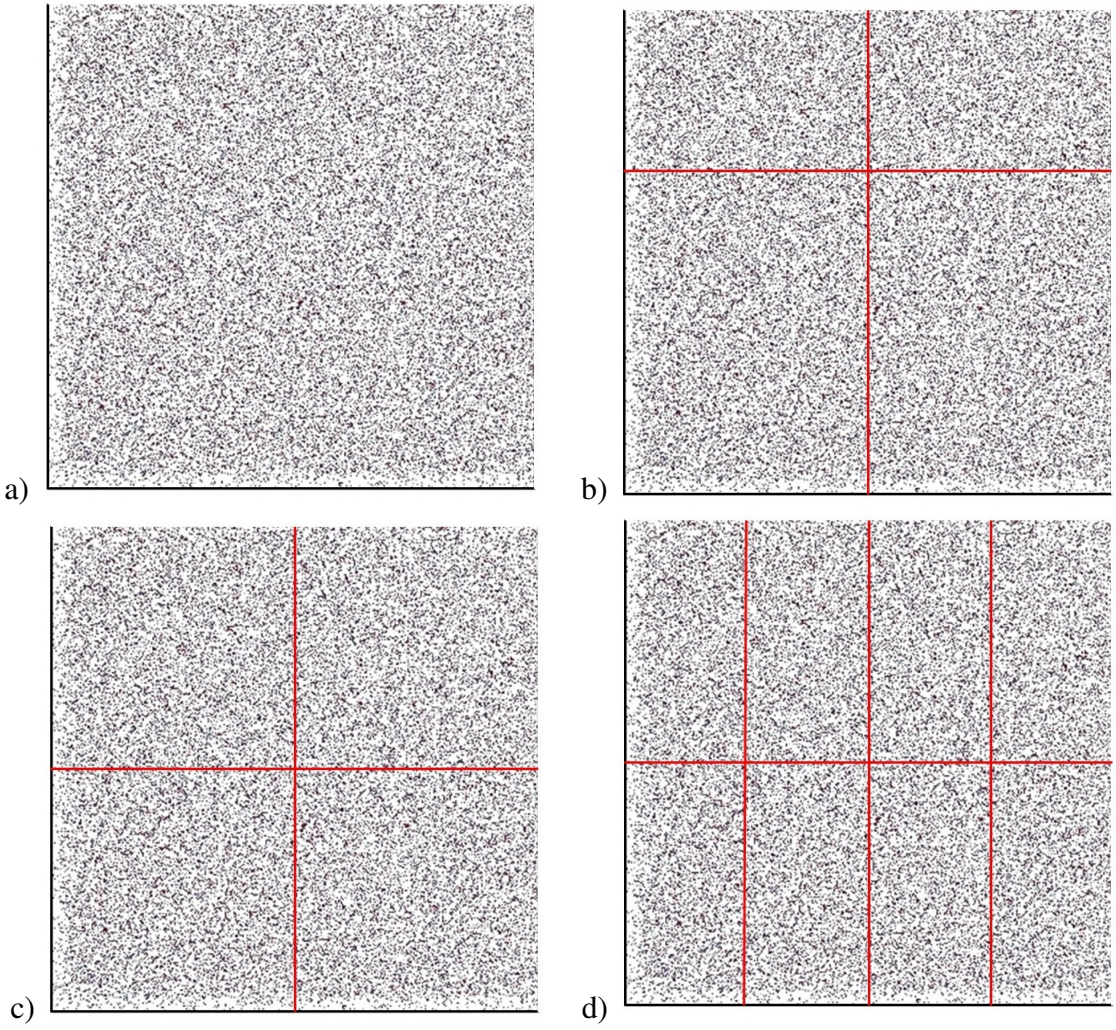

Figure 6.9: For a box filled with $250,000$ Gaseous Argon particles, comparison of the end state (Time $50ps$) among a) 1 processor, b) 8 processors, c) 16 processors, d) 32 processors.

Chapter 7

NUMERICAL EXAMPLE OF THE DEVELOPED PARALLEL SOLUTIONS

7.1 Nano Shock Wave

7.1.1 Description of the problem

A conservative system conforms to the conservation laws, namely, the conservation of mass, momentum and energy. For a steady shockwave, those three conservation laws apply across the shock front.[53,60,64,65]

Conversation of Mass

$$u_s \cdot \rho_0 = (u_s - u_p) \cdot \rho_1 \tag{7.1}$$

In which, ρ_0 is the density of un-shocked material, ρ_1 is the density of shocked material. u_s is the shockwave propagation speed, u_p is the piston speed. The conservation of mass is straightforward, namely, during the shock process, no mass gain or loss.

Conversation of Momentum

$$P - P_0 = u_p \cdot u_s \cdot \rho_0 = u_p \cdot (u_s - u_p) \cdot \rho_1 \tag{7.2}$$

In which, P is the pressure exerted on the outer wall of the piston, and P_0 is the pressure exerted on the un-shocked material. The conservation of the momentum dictates that the pressure exerted on the piston to drive it into the fluid are transformed into the increased momentum of the shocked material.

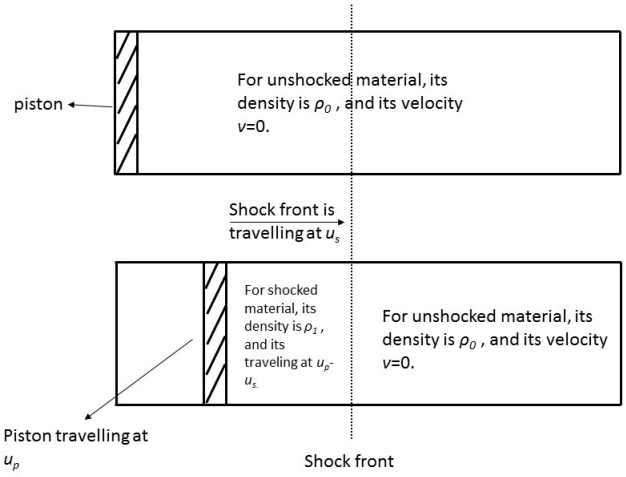

Figure 7.1: Scheme of a shock process.

Conversation of Energy

$$P \cdot u_p = \frac{1}{2} u_s \cdot \rho_0 \cdot u_p^2 + u_s \cdot \rho_0 \cdot (e - e_0) \tag{7.3}$$

In which, e and is e_0 are the internal energy of the shocked material and un-shocked material respectively. The energy applied to the piston is harnessed by the shocked material to increase its kinetic and potential energy.

For the purpose of convenience, a shock front frame is adopted in which

$$u_s = 0$$

as the observer is riding on the shock front, he will find the piston and the shocked material crushing into him in a velocity of

$$u = u_p - u_s$$

and the rest of the material (un-shocked) will also move toward himself in

$$u_0 = u_s$$

By substituting the relationship among u, u_0, u_p and u_s into eq.7.1-eq.7.3, the Hugoniot-Rankine relations are derived:

$$u \cdot \rho = u_0 \rho_0 \tag{7.4}$$

$$P+\rho\cdot u^2 = P_0+\rho_0\cdot u_0^2 \tag{7.5}$$

$$\frac{P}{\rho}+e+\frac{1}{2}u^2 = \frac{P_0}{\rho_0}+e_0+\frac{1}{2}u_0^2 \tag{7.6}$$

by eliminating the velocity items in eq.7.4-eq.7.6

$$e-e_0 = \frac{1}{2}(P+P_0)\cdot(\frac{1}{\rho_0}-\frac{1}{\rho}) \tag{7.7}$$

The Hugonoit equation describes a curve in which all points represent all the jumped states from a known initial state.

By incorporating mass and momentum equation eq.7.4 and eq.7.5:

$$\rho_0^2\cdot u_0^2 = \rho^2\cdot u^2 = -\frac{P-P_0}{(\frac{1}{\rho}-\frac{1}{\rho_0})} \tag{7.8}$$

This equation describes the Rayleigh line, which describes a single jump event between the initial state and the shocked state on the $\frac{1}{\rho}-P$ diagram.

Calculation of Local Pressure

For an ideal gas, its local pressure can be measured as global pressure, which means, only the particle-wall interaction is calculated. However, for dense gas, fluids and shockwave, the ideal gas assumption is no longer valid, particle-particle contact needs also be taken into consideration.[53,60,64,65,67]

Local pressure is important because it can be used to calculate interface tension and analyse the mechanical responses to strain, heat and phase transformations.[55,99] Lots of researches have been undertaken in the local pressure in fluids[60,64], polymers[166,165,120] and at surfaces[53,67], and etc.

In this study, the local pressure is calculated by:

$$P_{xx} = \sum_{i=1}^{N}\frac{1}{2}mv_i^2 + \sum_{i=j}^{N}\sum_{j=1}^{N}\frac{F_{ij}}{A} \tag{7.9}$$

in which, P_{xx} is the horizontal pressure, m is the particle mass, v is the velocity of the particle, N is the total number of particles in the simulation domain, F_{ij} is the intermolecular force acting on particle i, A is the longitudinal cross sectional area of the simulation domain. The first term on the right side is the pressure from the dynamic thermal random motion, the second term on the right is the pressure from the inter-particle collision.

In this study, the local pressure is defined as the horizontal pressure at a certain

x location from pair-wise particle interaction (two-body interaction).[161,75,120,142,166] For example in Figure 7.4, the local pressure at x_0 is calculated by first setting up an imaginary sample bin around x_0, with a width of $2a$. Then, only particle pairs that have each of the particles across the x_0 will be counted for their inter-particle potential that acts on the x_0 wall. The choice of a should be finitely small but also at the same time it needs to be big enough for meaningful statistics sampling (avoiding statistical fluctuations). In this study, $2a$ is chosen to be same as the cutoff radius of the argon interaction potential.

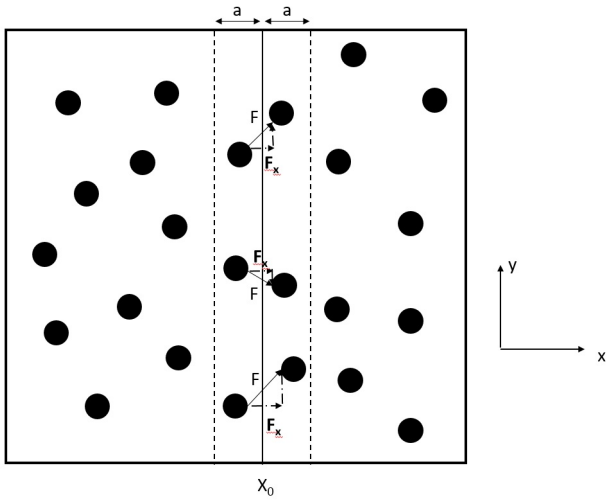

Figure 7.2: Selection of a sample bin.

7.1.2 Experiment setup

$10,201$ argon atoms are posited in a $2100\mathring{A} \times 2100\mathring{A}$ container. The equilibrium temperature is $107.3577K$, the number density of the atom is 0.00231315, and the interspacing of atoms is $20\mathring{A}$. Measured at after the system has reached equilibrium, the average velocity of the atom is $2.115\mathring{A}/ps$ ($211.5m/s$), and the system pressure is $8283996.2513Pa$ (81.76 atm).

First, 9 sets of simulations are conducted with different inlet wall velocities, with Mach number starting from just above the local sound speed and well above sound speed using 16 cores (4 rows and 4 columns). The simulation result is benchmarked with the theoretical results.

Second, an efficiency test across a range of multiple cores is set up. The system is scaled up to $250,000$ argon atoms in a $10100\mathring{A} \times 10100\mathring{A}$ box, the equilibrium temperature is kept at $50.011K$, the number density is 0.002402. A set of 5 simulations

are performed with wall velocity at $2.1\mathring{A}/ps$ on 1, 4, 32 and 64 cores. The runtime is 200 *ps*, this value is so chosen that for 64 cores (8 rows by 8 columns), no core will be left with no particles to handle and each processor will have approximately the same number of local particles.

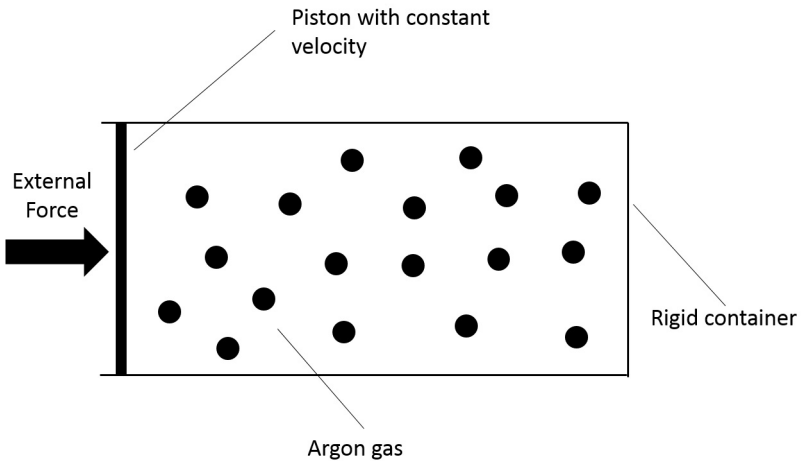

Figure 7.3: External force is applied on the planar wall on the left towards the right.

As is illustrated in Figure 7.3 and Figure 7.4, the left wall is treated as a piston that can move rightwards into the gas. The system is allowed 50 *ps* to reach equilibrium, then the piston will be driven rightwards in a constant speed, with a Mach number greater than 1.

After the system stabilizes, several physical variables are observed and collected to quantify the shockwave, e.g. local density, local pressure and etc. Those variables will then be popped into the theoretical framework to be benchmarked against the theoretical value.

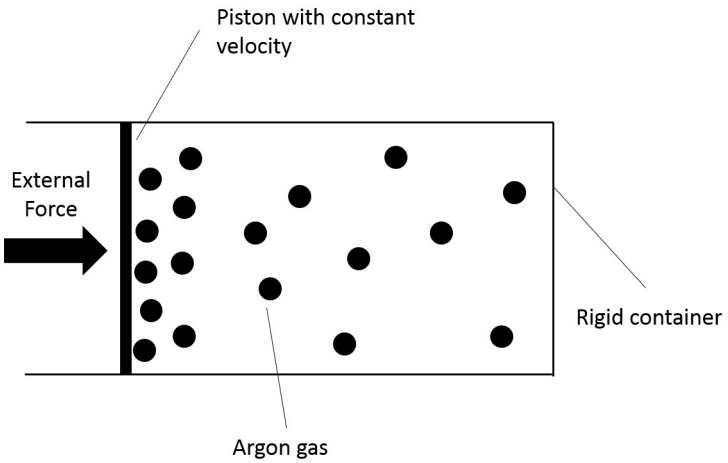

Figure 7.4: Particles on the left collide into each other as piston is driving at a shock-wave speed.

7.1.3 Results and Discussion

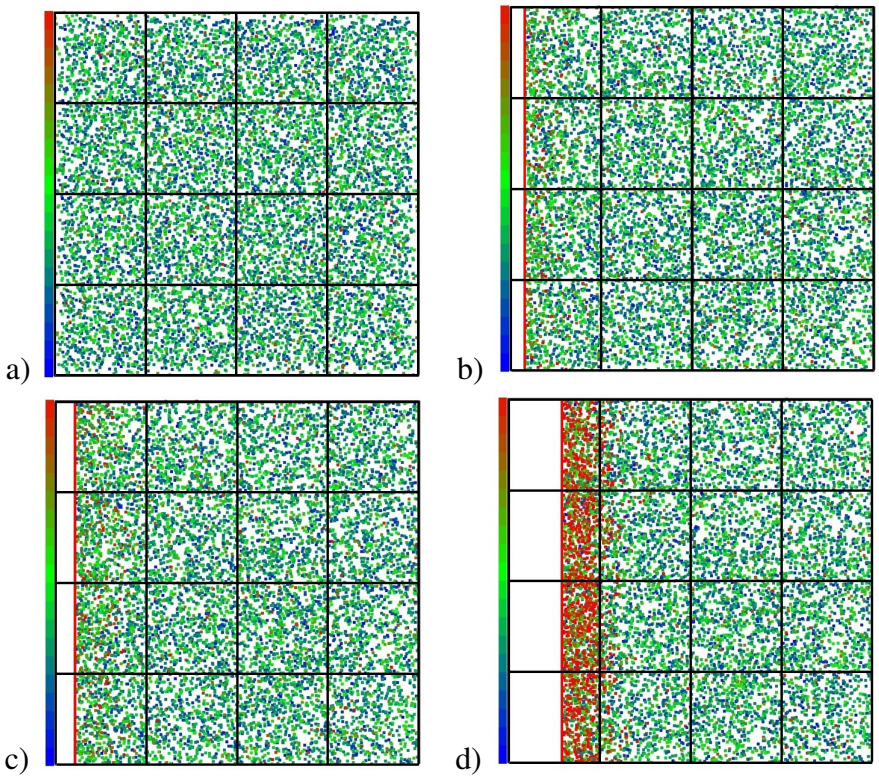

Figure 7.5: A motion sequence for a planar shockwave (Mach=1.66) in a box filled with 101,200 gaseous Argon particles executed on 4 processors. a) Time 0 *ps*, b) Time 30 *ps*, c) Time 40 *ps*, d) Time 90 *ps*.

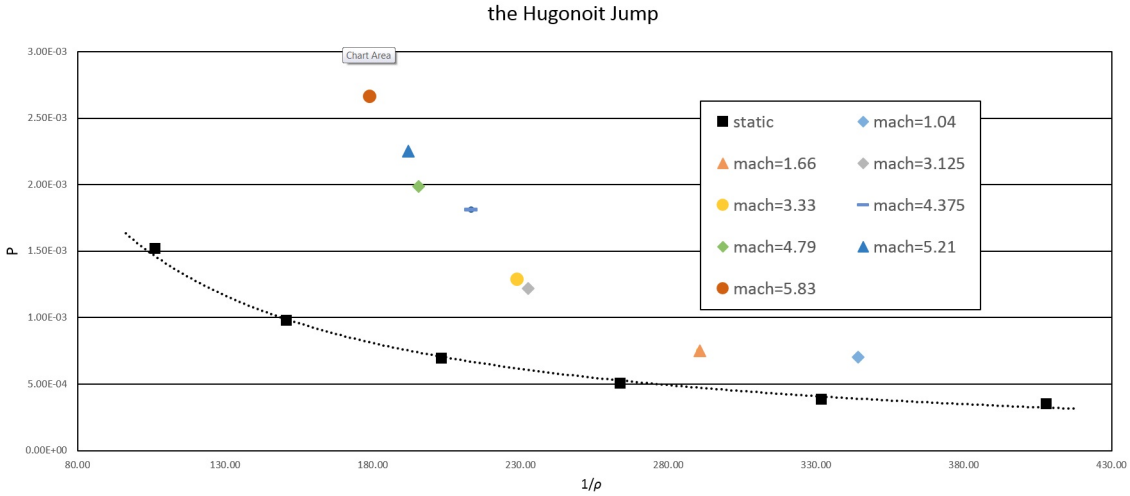

Figure 7.6: The Hugonoit jump.

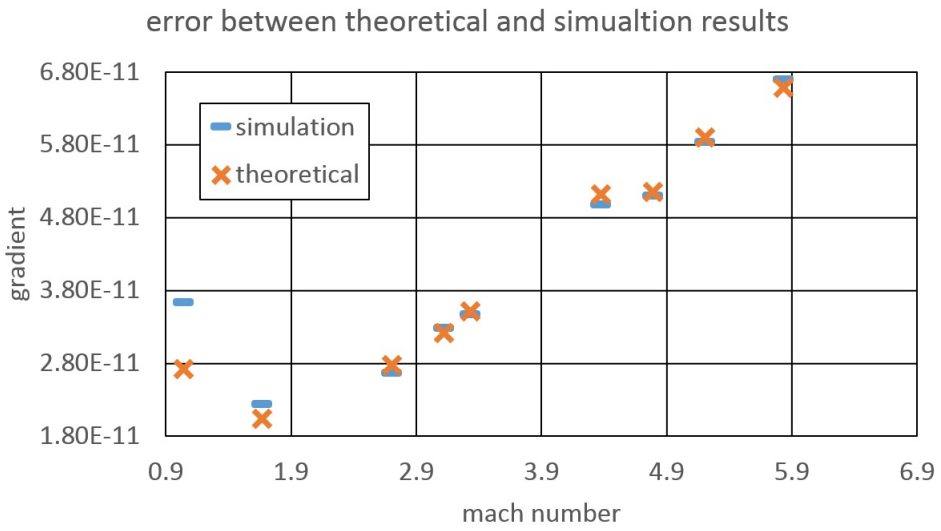

Figure 7.7: Comparison between theoretical value and simulations.

It could be seen from the particle motion evolution that as the left reflective wall drives into the right with a speed greater than the local speed of sound, it crashes into the

medium before the wave could make way for it. As the simulation progresses, more and more atoms accumulate on the driving wall side.

As can be seen from Figure 7.7, the error between the simulation and theoretical value decreases as the Mach number increases. The simulation is most inaccurate when the Mach number is between 1 to 1.5, the error margin reaches as high as 33.78% at Ma=1.04. However, from Ma=1.5 onwards, there is good agreement between the theoretical and simulation results, the error margin is controlled within 3, with an average of 2.02%.

In the second set of simulations, speedup from 8 to 64 processors are very good. The reason behind the good performance is that the nature of the shockwave determines that the medium doesn't have time to reflect the changes brought by the moving wall in the far end of the simulation domain. This means that even though the wall is driving into the box in supersonic speed creating a shock front where particles violently collide and stuck and move along with the wall, the rest of the system is completely unaware of this and thus remains a rarefied gas system. The speedup of a program is defined as follows:

$$S = \frac{t_s}{t_p} \tag{7.10}$$

in which t_s is the processing time for a sequential code, and t_p is the processing time of a parallelized code. Furthermore, t_p per processor could be calculated as the summation of computational time t_{cp} , halting (waiting) time t_{ht} resulting from load imbalance and communication time t_{cm} :

$$t_p = t_{cp} + t_{ht} + t_{cm} \tag{7.11}$$

Compared with a sequential code, in a parallel code

- With increasing number of processors P, each processors will have an increasingly smaller problem size, thus bringing down the t_{cp} ;
- the communication time t_{cm} scales with the perimeter of the box, it is evident that with bigger P comes smaller side of the box, but openning up channel could be huge expensive, and hence too many processors will translate into higher t_{cm}.
- The halting time t_{ht} is the time that faster processors will have to wait for the slower (heavier laboured) processors for synchronization (communication). The halting time might increase because of the high communication time in this particular case;

From the analysis above, it can be derived that t_p should most probably increase with increasing P , therefore the speedup should increase with negative gradient as the number of processors increases.

The super-linear performance in 64 cores might result from the fact that it has used approximately twice more memory on each node than the memory allocated to the 32 cores. The observed efficiency keeps picking up as the number of processors increases, this should most probably result from the implementation details of the HPC cluster, i.e. extra memory available for each cores.

It should be noted here that the parameters of the second set of simulations is designed to avoid significant load imbalance, but for a highly dynamic physics system as the planar shockwave, dynamic balance should be introduced to enhance the performance.

Core [-]	Time [s]	Speedup [-]	Efficiency [%]
1	6864	1	100
4	3637	1.89	47.18
8	1436	1.77	59.75
32	263	26.10	81.56
64	90	76.27	119.1667

Table 7.1: Calculated parallelisation efficiency for nano shockwave.

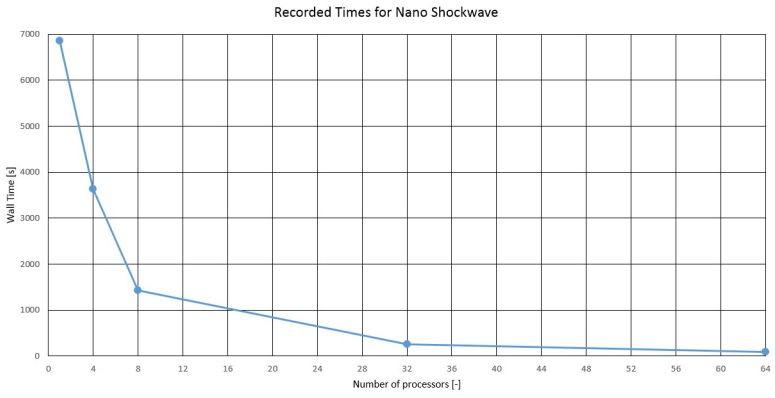

Figure 7.8: Recorded Times for Nano Shockwave in 1, 4, 8, 32, and 64 cores.

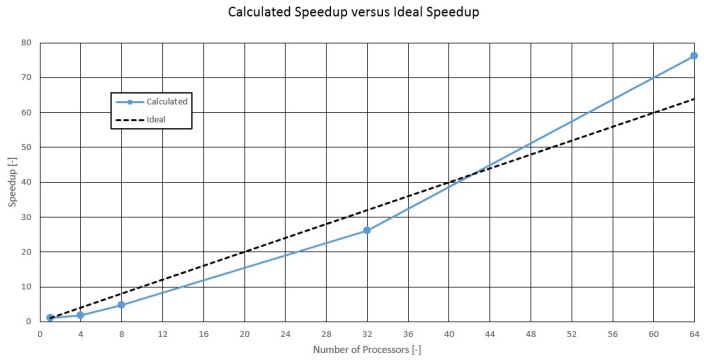

Figure 7.9: The calculated speedup against the ideal speedup for nano shockwave.

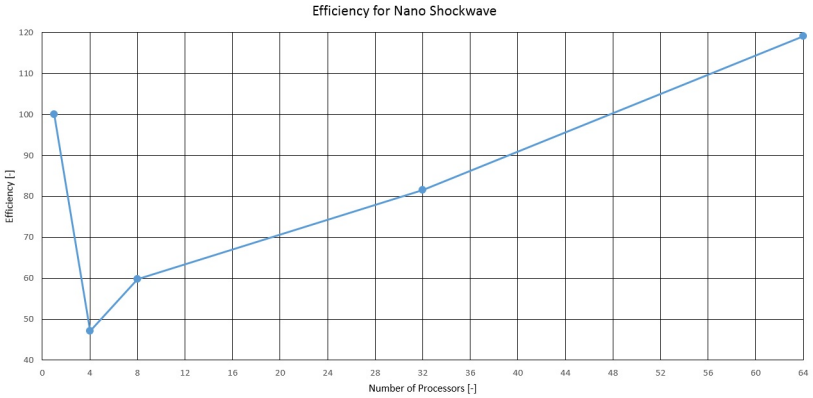

Figure 7.10: The efficiency of the nano shockwave.

From the results, it is reasonable to claim that Molecular Dynamics simulation can yield truthful results on a steady front shock wave.

Chapter 8

CONCLUSIONS AND FUTURE WORK

Three new implementations, a novel linear contact detection algorithm MR_PB for boundary conditions, share-memory based parallelization solutions and distributed-memory based parallelization solution, have been implemented into the open source in-house 2D Molecular Dynamics Solutions code YNANO. A summary of this thesis is presented and future research directions are suggested in this chapter.

Conclusions

In this thesis, a novel MR_PB algorithm, shared-memory based parallelization solutions of YNANO and distributed-memory based parallelization solutions of YNANO have been developed, validated and tested.

The MR_PB is shown to have a linear efficiency ($O(n)$) to the size of the problem. The developed novel sort/search algorithm offers an elegant solution to the periodic boundary conditions problem in MR_sort and search algorithm, and has greatly expanded the applicability of the original MR detection algorithm in area like Nano Aero-dynamics and Nano Fluid-dynamics whose nature of the problem might require a constant flow.

The developed shared-memory parallelization solution has greatly utilized the multi-core computers available in modern day computer labs and clusters, and enhanced the computational efficiency of YNANO. With the developed shared-memory parallelization solution, desktop computers could easily harness the full potential of its multi-cores in simulating systems with a huge number of particles.

The developed distributed-memory parallelization solution has been validated in systems with up to $250,000$ particles in up to 64 cores (according to the implementation details of the cluster used, the 64 cores are spread among at least 4 nodes). The parallel version of the code has also been shown to have a good efficiency and

scalability performance. The developed solution greatly removes the computational bottleneck of the YNANO MD simulation, and has prepared it to simulate systems with a larger number of particle and more complicated interaction models among them under a reasonable amount of time.

Future Work

Possibilities for further research are presented and discussed in the remainder of this chapter.

1. For the distributed-memory based parallelization solutions, performance of the parallelization solutions relies on stream-lining of the overhead communicational cost between processors. During the stage of migration, as all interfacial particles will need to be checked for its new position by default, only the outer (spatially speaking, the part close to the interfacial boundary) part of the internal particles need to be parsed and judged (due to the stability criteria). Since the current MR Sort algorithm is architected in a Cartesian way that cell list is constructed horizontally and extends vertically, all internal particles have to be parsed for the check. To improve future performance, it is suggested that future work designs a novel way construct the internal particle cell list. One possible solution is to build a radial-based cell list so that only the outer laps of the internal sub section needs to be parsed.

2. For the shared-memory based parallelization solutions, the current parallelization strategy keeps to the original procedure in YNANO in which MR search is directly followed by force calculation for each cell box. The advantage of this strategy is that the code doesn't need to hold a lot of RAM to store particle pair information during the contact search processor for the force calculation process later. For homogeneous problems, performance shouldn't be a problem, but for heterogeneous problems where some parts of the simulation domain may have more contact pairs than elsewhere, it could happen that some processors will need to sit in idle to wait for other processors finishing their workload. This is a huge waste of computational resources. Performance could enhance greatly if these two takes are separated, and after the accomplishment of the first task, work load would be re-calculated and re-distributed across processors. However, an obvious problem to the proposed solution is how to deal with the vast memory required to store the contact pair information. This invites more careful analysis of the simulation problem and data structure design. As can be seen in the shared-memory systems, load balancing is the key to optimal parallelization.

3. The wide application of YNANO suggests that not all simulation problems are homogeneous. Future work should also include implementation of the load balancing to ensure an equal workload on all processor to for better utilization of computational power and to reduce the idle time. The implementation of dynamic load balancing would take place right after particle migration, processors will communicate with each other how many local particles each is handling, and workload will be re-distributed roughly evenly. It could be expected that dynamic load balancing would disturb the book-keeping of the spatial relation of cell list upon which the MR contact detection is built, therefore future work should look into how to reconcile this conflict.[25, 10, 40, 41, 47, 105, 2, 1, 121, 8, 69]

4. Also, the developed MR_PB contact detection algorithm, shared-memory system based and distributed-memory system based solutions are designed for $2D$ version of the YNANO, and it would be highly desirable if they are implemented in the $3D$ version of the YNANO as well. Implementing the same features on a $3D$ system should be relatively straightforward, the $3D$ version of the MR_PB is already readily available as discussed in the thesis, the shared-memory based parallelization solutions would require more careful examination into how to define the contact mask and how to initialize the pointers to the transitional particle list. The distributed-memory based parallelization solutions would invite more spatial classification of the particles and as well as the $2D$ to $3D$ communication strategy.

5. Finally, implementing MR_PB algorithm developed in this thesis onto the parallelized versions of the YNANO would open up possibility to simulate a range of more complicate and dynamic system (for instance Nano Fluid-dynamics problems, as the Lattice Boltzman (LB) has been already coupled with DEM by Feng et al.[48] in order to solve fluid-particle interactions.) The solution should also be relatively straightforward, it would require communications among the processors on each side during force transmission stage and the migration stage.

References

[1] ANDOH, Y. ; YOSHII, N. ; FUJIMOTO, K. ; MIZUTANI, K. ; KOJIMA, H. ; YAMADA, A. ; OKAZAKI, S. ; KAWAGUCHI, K. ; NAGAO, H. ; IWAHASHI, K. u. a.: MODYLAS: a highly parallelized general-purpose molecular dynamics simulation program for large-scale systems with long-range forces calculated by Fast Multipole Method (FMM) and highly scalable fine-grained new parallel processing algorithms. In: *Journal of Chemical Theory and Computation* 9 (2013), Nr. 7, S. 3201–3209

[2] ANDOH, Y. ; YOSHII, N. ; FUJIMOTO, K. ; MIZUTANI, K. ; KOJIMA, H. ; YAMADA, A. ; OKAZAKI, S. ; KAWAGUCHI, K. ; NAGAO, H. ; IWAHASHI, K. u. a.: MODYLAS: a highly parallelized general-purpose molecular dynamics simulation program for large-scale systems with long-range forces calculated by Fast Multipole Method (FMM) and highly scalable fine-grained new parallel processing algorithms. In: *Journal of Chemical Theory and Computation* 9 (2013), Nr. 7, S. 3201–3209

[3] BAI, M. ; SUN, S. ; TANG, H. ; DOU, Y. ; LO, G. V.: An SPMD-Like Algorithm for Parallelizing Molecular Dynamics Using OpenMP. In: *Computing in Science & Engineering* 15 (2013), Nr. 4, S. 48–56

[4] BAI, S.-r. ; RAN, L.-p. ; LU, K.-l. : Parallelization and performance tuning of molecular dynamics code with OpenMP. In: *Journal of Central South University of Technology* 13 (2006), Nr. 3, S. 260–264

[5] BATHE, K.-J. ; WILSON, E. L.: Numerical methods in finite element analysis. (1976)

[6] BEAZLEY, D. M. ; LOMDAHL, P. S.: Message-passing multi-cell molecular dynamics on the Connection Machine 5. In: *Parallel Computing* 20 (1994), Nr. 2, S. 173–195

[7] BERENDSEN, H. J. ; SPOEL, D. van d. ; DRUNEN, R. van: GROMACS: A message-passing parallel molecular dynamics implementation. In: *Computer Physics Communications* 91 (1995), Nr. 1, S. 43–56

[8] BHANDARKAR, M. ; KALÉ, L. V. ; STURLER, E. de ; HOEFLINGER, J. : Adaptive load balancing for MPI programs. In: *Computational Science-ICCS 2001*. Springer, 2001, S. 108–117

[9] BIRD, G. A.: Molecular gas dynamics and the direct simulation of gas flows. (1994)

[10] BOILLAT, J. ; BRUGE, F. ; KROPF, P. : A dynamic load-balancing algorithm for molecular dynamics simulation on multi-processor systems. In: *Journal of computational physics* 96 (1991), Nr. 1, S. 1–14

[11] BORŠTNIK, U. ; MILLER, B. T. ; BROOKS, B. R. ; JANEŽIČ, D. : The distributed diagonal force decomposition method for parallelizing molecular dynamics simulations. In: *Journal of computational chemistry* 32 (2011), Nr. 14, S. 3005–3013

[12] BORŠTNIK, U. ; MILLER, B. T. ; BROOKS, B. R. ; JANEŽIČ, D. : Implementation of the force decomposition machine for molecular dynamics simulations. In: *Journal of Molecular Graphics and Modelling* 38 (2012), S. 243–247

[13] BOWERS, K. J. ; CHOW, E. ; XU, H. ; DROR, R. O. ; EASTWOOD, M. P. ; GREGERSEN, B. ; KLEPEIS, J. L. ; KOLOSSVARY, I. ; MORAES, M. ; SACERDOTI, F. D. u. a.: Scalable algorithms for molecular dynamics simulations on commodity clusters. In: *SC 2006 Conference, Proceedings of the ACM/IEEE* IEEE, 2006, S. 43–43

[14] BOWERS, K. J. ; DROR, R. O. ; SHAW, D. E.: Overview of neutral territory methods for the parallel evaluation of pairwise particle interactions. In: *Journal of Physics: Conference Series* Bd. 16 IOP Publishing, 2005, S. 300

[15] BOWERS, K. J. ; DROR, R. O. ; SHAW, D. E.: The midpoint method for parallelization of particle simulations. In: *The Journal of chemical physics* 124 (2006), Nr. 18, S. 184109

[16] BOWERS, K. J. ; DROR, R. O. ; SHAW, D. E.: Zonal methods for the parallel execution of range-limited N-body simulations. In: *Journal of Computational Physics* 221 (2007), Nr. 1, S. 303–329

[17] BOYD, I. ; SUN, Q. ; MARTIN, M. : Simulation of Micro-Scale Aerodynamics. In: *Ann Arbor* 1001 (2003), S. 48109

[18] BROOKS, B. R. ; BROOKS, C. L. ; MACKERELL, A. D. ; NILSSON, L. ; PETRELLA, R. J. ; ROUX, B. ; WON, Y. ; ARCHONTIS, G. ; BARTELS, C. ; BORESCH, S. u. a.: CHARMM: the biomolecular simulation program. In: *Journal of computational chemistry* 30 (2009), Nr. 10, S. 1545–1614

[19] BROOKS, B. R. ; HODOŠČEK, M. : *Parallelization of CHARMM for MIMD machines*. 1992

[20] BROWN, D. ; CLARKE, J. H. ; OKUDA, M. ; YAMAZAKI, T. : A domain decomposition parallelization strategy for molecular dynamics simulations on distributed memory machines. In: *Computer Physics Communications* 74 (1993), Nr. 1, S. 67–80

[21] BROWN, D. ; CLARKE, J. H. ; OKUDA, M. ; YAMAZAKI, T. : A domain decomposition parallelization strategy for molecular dynamics simulations on distributed memory machines. In: *Computer Physics Communications* 74 (1993), Nr. 1, S. 67–80

[22] BROWN, D. ; CLARKE, J. H. ; OKUDA, M. ; YAMAZAKI, T. : A domain decomposition parallel processing algorithm for molecular dynamics simulations of polymers. In: *Computer Physics Communications* 83 (1994), Nr. 1, S. 1–13

[23] BROWN, W. M. ; YAMADA, M. : Implementing molecular dynamics on hybrid high performance computers Three body potentials. In: *Computer Physics Communications* 184 (2013), Nr. 12, S. 2785–2793

[24] BRUGE, F. : A mixed geometric-systolic approach to parallel molecular dynamics simulations. In: *Computer physics communications* 90 (1995), Nr. 1, S. 59–65

[25] BRUGE, F. ; FORNILI, S. : A distributed dynamic load balancer and its implementation on multi-transputer systems for molecular dynamics simulation. In: *Computer Physics Communications* 60 (1990), Nr. 1, S. 39–45

[26] BUCHHOLZ, M. ; BUNGARTZ, H.-J. ; VRABEC, J. : Software design for a highly parallel molecular dynamics simulation framework in chemical engineering. In: *Journal of Computational Science* 2 (2011), Nr. 2, S. 124–129

[27] CHAPLOT, S. : Parallelization in classical molecular dynamics simulation and applications. In: *Computational materials science* 37 (2006), Nr. 1, S. 146–151

[28] CHAPMAN, B. ; JOST, G. ; VAN DER PAS, R. : *Using OpenMP: portable shared memory parallel programming*. Bd. 10. MIT press, 2008

[29] CHEN X, Y. J. CHAN A H C C. CHAN A H C: A case study of impact on glass using the combined Finite-Discrete Element method. In: *Discrete Element Methods, Simulatios of Discontinua: Theory and Application* (2010)

[30] CHORLEY, M. J. ; WALKER, D. W.: Performance analysis of a hybrid MPI/OpenMP application on multi-core clusters. In: *Journal of Computational Science* 1 (2010), Nr. 3, S. 168–174

[31] CHORLEY, M. J. ; WALKER, D. W. ; GUEST, M. F.: Hybrid message-passing and shared-memory programming in a molecular dynamics application on multicore clusters. In: *International Journal of High Performance Computing Applications* (2009)

[32] CLARK, T. W. ; HANXLEDEN, R. V. ; MCCAMMON, J. A. ; SCOTT, L. R.: Parallelizing molecular dynamics using spatial decomposition. In: *Scalable High-Performance Computing Conference, 1994., Proceedings of the* IEEE, 1994, S. 95–102

[33] CLARK, T. W. ; HANXLEDEN, R. V. ; MCCAMMON, J. A. ; SCOTT, L. R.: Parallelizing molecular dynamics using spatial decomposition. In: *Scalable High-Performance Computing Conference, 1994., Proceedings of the* IEEE, 1994, S. 95–102

[34] CLARK, T. W. ; MCCAMMON, J. A.: Parallelization of a molecular dynamics non-bonded force algorithm for MIMD architecture. In: *Computers & Chemistry* 14 (1990), Nr. 3, S. 219–224

[35] CONSOLINI, L. ; AGGARWAL, S. K. ; MURAD, S. : A molecular dynamics simulation of droplet evaporation. In: *International journal of heat and mass transfer* 46 (2003), Nr. 17, S. 3179–3188

[36] COUTURIER, R. ; CHIPOT, C. : Parallel molecular dynamics using OpenMP on a shared memory machine. In: *Computer physics communications* 124 (2000), Nr. 1, S. 49–59

[37] CUMMINGS, P. T.: Molecular simulation of complex systems using massively parallel supercomputers. In: *Fluid phase equilibria* 144 (1998), Nr. 1, S. 331–342

[38] DALBANO, S. ; GONZALO, G. : *Computational and Algorithmic Solutions for Large Scale Combined Finite-Discrete Elements Simulations.*, Queen Mary University of London, Diss., 2014

[39] DEBOLT, S. E. ; KOLLMAN, P. A.: AMBERCUBE MD, parallelization of Amber's molecular dynamics module for distributed-memory hypercube computers. In: *Journal of Computational Chemistry* 14 (1993), Nr. 3, S. 312–329

[40] DENG, Y.-F. ; MCCOY, R. A. ; MARR, R. B. ; PEIERLS, R. F. ; YASAR, O. : Molecular dynamics on distributed-memory MIMD computers with load balancing. In: *Applied mathematics letters* 8 (1995), Nr. 3, S. 37–41

[41] DENG, Y. ; PEIERLS, R. F. ; RIVERA, C. : An adaptive load balancing method for parallel molecular dynamics simulations. In: *Journal of Computational Physics* 161 (2000), Nr. 1, S. 250–263

[42] DI COLA, D. ; DERIU, A. ; SAMPOLI, M. ; TORCINI, A. : Proton dynamics in supercooled water by molecular dynamics simulations and quasielastic neutron scattering. In: *The Journal of chemical physics* 104 (1996), Nr. 11, S. 4223–4232

[43] EISENHAUER, G. ; SCHWAN, K. : Design and analysis of a parallel molecular dynamics application. In: *Journal of Parallel and Distributed Computing* 35 (1996), Nr. 1, S. 76–90

[44] EISENHAUER, G. ; SCHWAN, K. : Design and analysis of a parallel molecular dynamics application. In: *Journal of Parallel and Distributed Computing* 35 (1996), Nr. 1, S. 76–90

[45] ELMO, D. ; VYAZMENSKY, A. ; STEAD, D. ; RANCE, J. u. a.: A hybrid FEM/DEM approach to model the interaction between open-pit and underground block-caving mining. In: *1st Canada-US Rock Mechanics Symposium* American Rock Mechanics Association, 2007

[46] FANG, T.-H. ; WENG, C.-I. : Three-dimensional molecular dynamics analysis of processing using a pin tool on the atomic scale. In: *Nanotechnology* 11 (2000), Nr. 3, S. 148

[47] FATTEBERT, J.-L. ; RICHARDS, D. F. ; GLOSLI, J. N.: Dynamic load balancing algorithm for molecular dynamics based on Voronoi cells domain decompositions. In: *Computer Physics Communications* 183 (2012), Nr. 12, S. 2608–2615

[48] FENG, Y. ; HAN, K. ; OWEN, D. : Coupled lattice Boltzmann method and discrete element modelling of particle transport in turbulent fluid flows: Computational issues. In: *International Journal for Numerical Methods in Engineering* 72 (2007), Nr. 9, S. 1111

[49] FINCHAM, D. : Parallel computers and molecular simulation. In: *Molecular Simulation* 1 (1987), Nr. 1-2, S. 1–45

[50] FINCHAM, D. : Leapfrog rotational algorithms. In: *Molecular Simulation* 8 (1992), Nr. 3-5, S. 165–178

[51] FINCHAM, D. ; RALSTON, B. : Molecular dynamics simulation using the CRAY-1 vector processing computer. In: *Computer physics communications* 23 (1981), Nr. 2, S. 127–134

[52] FOREST, E. ; RUTH, R. D.: Fourth-order symplectic integration. In: *Physica D: Nonlinear Phenomena* 43 (1990), Nr. 1, S. 105–117

[53] FORET, L. ; KÜHN, R. ; WÜRGER, A. : Disjoining pressure of discrete surface charges on thin aqueous films. In: *Physical review letters* 89 (2002), Nr. 15, S. 156102

[54] FOX, G. C.: Solving problems on concurrent processors. (1988)

[55] FRENKEL, D. ; SMIT, B. : *Understanding molecular simulation: from algorithms to applications*. Bd. 1. Academic press, 2001

[56] GOEDECKER, S. : Optimization and parallelization of a force field for silicon using OpenMP. In: *Computer physics communications* 148 (2002), Nr. 1, S. 124–135

[57] GOGA, N. ; MARRINK, S. ; CIOROMELA, R. ; MOLDOVEANU, F. : GPU-SD and DPD parallelization for Gromacs tools for molecular dynamics simulations. In: *Bioinformatics & Bioengineering (BIBE), 2012 IEEE 12th International Conference on* IEEE, 2012, S. 251–254

[58] GOLDSTEIN, D. ; HANDLER, R. ; SIROVICH, L. : Modeling a no-slip flow boundary with an external force field. In: *Journal of Computational Physics* 105 (1993), Nr. 2, S. 354–366

[59] GREENWELL, D. ; KALIA, R. K. ; PATTERSON, J. ; VASHISHTA, P. : Molecular Dynamics Algorithm on the connection machine. In: *International Journal of High Speed Computing* 1 (1989), Nr. 02, S. 321–328

[60] GRONBECH-JENSEN, N. ; HUMMER, G. ; BEARDMORE, K. M.: Lekner summation of Coulomb interactions in partially periodic systems. In: *Molecular Physics* 92 (1997), Nr. 5, S. 941–946

[61] GROPP, W. ; LUSK, E. ; SKJELLUM, A. : *Using MPI Portable Parallel Programming with Message-Passing Interface. second*. 1999

[62] GRUBMÜLLER, H. ; HELLER, H. ; WINDEMUTH, A. ; SCHULTEN, K. : Generalized Verlet algorithm for efficient molecular dynamics simulations with long-range interactions. In: *Molecular Simulation* 6 (1991), Nr. 1-3, S. 121–142

[63] GRUBMÜLLER, H. ; HELLER, H. ; WINDEMUTH, A. ; SCHULTEN, K. : Generalized Verlet algorithm for efficient molecular dynamics simulations with long-range interactions. In: *Molecular Simulation* 6 (1991), Nr. 1-3, S. 121–142

[64] HAFSKJOLD, B. ; IKESHOJI, T. : Microscopic pressure tensor for hard-sphere fluids. In: *Physical Review E* 66 (2002), Nr. 1, S. 011203

[65] HARASIMA, A. : Molecular theory of surface tension. In: *Adv. Chem. Phys* 1 (1958), S. 203–237

[66] HEFFELFINGER, G. S.: Parallel atomistic simulations. In: *Computer physics communications* 128 (2000), Nr. 1, S. 219–237

[67] HEINZ, H. ; PAUL, W. ; SUTER, U. W. ; BINDER, K. : Analysis of the phase transitions in alkyl-mica by density and pressure profiles. In: *The Journal of chemical physics* 120 (2004), Nr. 8, S. 3847–3854

[68] HELLER, H. ; GRUBMÜLLER, H. ; SCHULTEN, K. : Molecular dynamics simulation on a parallel computer. In: *Molecular simulation* 5 (1990), Nr. 3-4, S. 133–165

[69] HESS, B. ; KUTZNER, C. ; VAN DER SPOEL, D. ; LINDAHL, E. : GROMACS 4: algorithms for highly efficient, load-balanced, and scalable molecular simulation. In: *Journal of chemical theory and computation* 4 (2008), Nr. 3, S. 435–447

[70] HILBERS, P. ; ESSELINK, K. : Parallel computing and molecular dynamics simulations. In: *Computer Simulation in Chemical Physics*. Springer, 1993, S. 473–495

[71] HOSHINO, T. ; TAKENOUCHI, K. : Processing of the molecular dynamics model by the parallel computer PAX. In: *Computer physics communications* 31 (1984), Nr. 4, S. 287–296

[72] HU, Y. C. ; LU, H. ; COX, A. L. ; ZWAENEPOEL, W. : OpenMP for networks of SMPs. In: *Parallel Processing, 1999. 13th International and 10th Symposium on Parallel and Distributed Processing, 1999. 1999 IPPS/SPDP. Proceedings* IEEE, 1999, S. 302–310

[73] HWANG, Y.-S. ; DAS, R. ; SALTZ, J. H. ; HODOSCEK, M. ; BROOKS, B. R.: Parallelizing molecular dynamics programs for distributed-memory machines. In: *Computing in Science & Engineering* (1995), Nr. 2, S. 18–29

[74] HWANG, Y.-S. ; DAS, R. ; SALTZ, J. H. ; HODOSCEK, M. ; BROOKS, B. R.: Parallelizing molecular dynamics programs for distributed-memory machines. In: *Computing in Science & Engineering* (1995), Nr. 2, S. 18–29

[75] IRVING, J. ; KIRKWOOD, J. G.: The statistical mechanical theory of transport processes. IV. The equations of hydrodynamics. In: *The Journal of chemical physics* 18 (1950), Nr. 6, S. 817–829

[76] JABBARZADEH, A. ; ATKINSON, J. ; TANNER, R. : Parallel simulation of shear flow of polymers between structured walls by molecular dynamics simulation on PVM. In: *Computer physics communications* 107 (1997), Nr. 1, S. 123–136

[77] JANAK, J. ; PATTNAIK, P. : Protein calculations on parallel processors. ii. parallel algorithm for the forces and molecular dynamics. In: *Journal of computational chemistry* 13 (1992), Nr. 9, S. 1098–1102

[78] JENSEN, R. P. ; BOSSCHER, P. J. ; PLESHA, M. E. ; EDIL, T. B.: DEM simulation of granular media structure interface: effects of surface roughness and

particle shape. In: *International Journal for Numerical and Analytical Methods in Geomechanics* 23 (1999), Nr. 6, S. 531–547

[79] JUNG, J. ; MORI, T. ; SUGITA, Y. : Midpoint cell method for hybrid (MPI+ OpenMP) parallelization of molecular dynamics simulations. In: *Journal of computational chemistry* 35 (2014), Nr. 14, S. 1064–1072

[80] KALÉ, L. ; SKEEL, R. ; BHANDARKAR, M. ; BRUNNER, R. ; GURSOY, A. ; KRAWETZ, N. ; PHILLIPS, J. ; SHINOZAKI, A. ; VARADARAJAN, K. ; SCHULTEN, K. : NAMD2: greater scalability for parallel molecular dynamics. In: *Journal of Computational Physics* 151 (1999), Nr. 1, S. 283–312

[81] KALIA, R. K. ; LEEUW, S. de ; NAKANO, A. ; VASHISHTA, P. : Molecular-dynamics simulations of Coulombic systems on distributed-memory MIMD machines. In: *Computer physics communications* 74 (1993), Nr. 3, S. 316–326

[82] KARAKASIDIS, T. E. ; CHOLEVAS, N. ; LIAKOPOULOS, A. : Parallel short range molecular dynamics simulations on computer clusters: Performance evaluation and modeling. In: *Mathematical and computer modelling* 42 (2005), Nr. 7, S. 783–798

[83] KARNIADAKIS, G. E. ; KIRBY II, R. M.: *Parallel scientific computing in C++ and MPI: a seamless approach to parallel algorithms and their implementation*. Bd. 1. Cambridge University Press, 2003

[84] KLEPEIS, J. L. ; LINDORFF-LARSEN, K. ; DROR, R. O. ; SHAW, D. E.: Long-timescale molecular dynamics simulations of protein structure and function. In: *Current opinion in structural biology* 19 (2009), Nr. 2, S. 120–127

[85] KOMANDURI, R. ; CHANDRASEKARAN, N. ; RAFF, L. : Effect of tool geometry in nanometric cutting: a molecular dynamics simulation approach. In: *Wear* 219 (1998), Nr. 1, S. 84–97

[86] KOMANDURI, R. ; CHANDRASEKARAN, N. ; RAFF, L. : MD simulation of indentation and scratching of single crystal aluminum. In: *Wear* 240 (2000), Nr. 1, S. 113–143

[87] KOMEIJI, Y. ; HARAGUCHI, M. ; NAGASHIMA, U. : Parallel molecular dynamics simulation of a protein. In: *Parallel computing* 27 (2001), Nr. 8, S. 977–987

[88] KOPLIK, J. ; BANAVAR, J. R. ; WILLEMSEN, J. F.: Molecular dynamics of Poiseuille flow and moving contact lines. In: *Physical review letters* 60 (1988), Nr. 13, S. 1282

[89] KUNASETH, M. ; RICHARDS, D. F. ; GLOSLI, J. N. ; KALIA, R. K. ; NAKANO, A. ; VASHISHTA, P. : Analysis of scalable data-privatization threading algorithms for hybrid MPI/OpenMP parallelization of molecular dynamics. In: *The Journal of Supercomputing* 66 (2013), Nr. 1, S. 406–430

[90] LA PENNA, G. ; MINICOZZI, V. ; MORANTE, S. ; ROSSI, G. ; SALINA, G. : Molecular Dynamics with the massively parallel APE computers. In: *Computer physics communications* 106 (1997), Nr. 1, S. 53–68

[91] LAKSHMINARASIMHULU, P. ; MADURA, J. D.: A cell multipole based domain decomposition algorithm for molecular dynamics simulation of systems of arbitrary shape. In: *Computer physics communications* 144 (2002), Nr. 2, S. 141–153

[92] LAUGA, E. ; BRENNER, M. ; STONE, H. : Microfluidics: the no-slip boundary condition. In: *Springer handbook of experimental fluid mechanics* (2007), S. 1219–1240

[93] LEI, Z. ; ROUGIER, E. ; KNIGHT, E. ; MUNJIZA, A. u. a.: Block Caving Induced Instability Analysis using FDEM. In: *47th US Rock Mechanics/Geomechanics Symposium* American Rock Mechanics Association, 2013

[94] LIEM, S. ; BROWN, D. ; CLARKE, J. H.: Molecular dynamics simulations on distributed memory machines. In: *Computer Physics Communications* 67 (1991), Nr. 2, S. 261–267

[95] LIN, S. ; MELLOR-CRUMMEY, J. ; PETTITT, B. M. ; PHILLIPS, G. : Molecular dynamics on a distributed-memory multiprocessor. In: *Journal of Computational Chemistry* 13 (1992), Nr. 8, S. 1022–1035

[96] LIU, G.-x. ; TAN, J.-z. ; NIU, C.-y. ; SHEN, J.-h. ; LUO, X.-m. ; SHEN, X. ; CHEN, K.-x. ; JIANG, H.-l. : Molecular dynamics simulations of interaction between protein-tyrosine phosphatase 1B and a bidentate inhibitor. In: *Acta Pharmacologica Sinica* 27 (2006), Nr. 1, S. 100–110

[97] MATSUNAGA, S. : Structural study on liquid Au–Cs alloys by computer simulations. In: *Journal of the Physical Society of Japan* 69 (2000), Nr. 6, S. 1712–1716

[98] MATTHEY, T. ; IZAGUIRRE, J. A.: ProtoMol: A Molecular Dynamics Framework with Incremental Parallelization. In: *PPSC*, 2001

[99] MCLENNAN, J. A.: *Introduction to nonequilibrium statistical mechanics*. Prentice Hall, 1989

[100] MEYER, R. : Efficient parallelization of short-range molecular dynamics simulations on many-core systems. In: *Physical Review E* 88 (2013), Nr. 5, S. 053309

[101] MITCHELL, P. ; FINCHAM, D. : Multicomputer molecular dynamics. In: *Future Generation Computer Systems* 9 (1993), Nr. 1, S. 5–10

[102] MÜLLER-PLATHE, F. : Parallelising a molecular dynamics algorithm on a multi-processor workstation. In: *Computer Physics Communications* 61 (1990), Nr. 3, S. 285–293

[103] MÜLLER-PLATHE, F. : Parallelising a molecular dynamics algorithm on a multi-processor workstation. In: *Computer Physics Communications* 61 (1990), Nr. 3, S. 285–293

[104] MURTY, R. ; OKUNBOR, D. : Efficient parallel algorithms for molecular dynamics simulations. In: *Parallel Computing* 25 (1999), Nr. 3, S. 217–230

[105] NAKANO, A. ; CAMPBELL, T. : An adaptive curvilinear-coordinate approach to dynamic load balancing of parallel multiresolution molecular dynamics. In: *Parallel Computing* 23 (1997), Nr. 10, S. 1461–1478

[106] NAKANO, A. ; KALIA, R. K. ; VASHISHTA, P. : Multiresolution molecular dynamics algorithm for realistic materials modeling on parallel computers. In: *Computer Physics Communications* 83 (1994), Nr. 2, S. 197–214

[107] NELSON, M. T. ; HUMPHREY, W. ; GURSOY, A. ; DALKE, A. ; KALÉ, L. V. ; SKEEL, R. D. ; SCHULTEN, K. : NAMD: a parallel, object-oriented molecular dynamics program. In: *International Journal of High Performance Computing Applications* 10 (1996), Nr. 4, S. 251–268

[108] NICOL, D. M. ; SALTZ, J. H.: Dynamic remapping of parallel computations with varying resource demands. In: *Computers, IEEE Transactions on* 37 (1988), Nr. 9, S. 1073–1087

[109] NISHIURA, D. ; SAKAGUCHI, H. : Parallel-vector algorithms for particle simulations on shared-memory multiprocessors. In: *Journal of Computational Physics* 230 (2011), Nr. 5, S. 1923–1938

[110] NOMURA, K.-i. ; KALIA, R. K. ; NAKANO, A. ; VASHISHTA, P. : A scalable parallel algorithm for large-scale reactive force-field molecular dynamics simulations. In: *Computer Physics Communications* 178 (2008), Nr. 2, S. 73–87

[111] NYLAND, L. ; PRINS, J. ; YUN, R. H. ; HERMANS, J. ; KUM, H.-C. ; WANG, L. : Achieving scalable parallel molecular dynamics using dynamic spatial domain decomposition techniques. In: *Journal of Parallel and Distributed Computing* 47 (1997), Nr. 2, S. 125–138

[112] OGATA, S. ; LIDORIKIS, E. ; SHIMOJO, F. ; NAKANO, A. ; VASHISHTA, P. ; KALIA, R. K.: Hybrid finite-element/molecular-dynamics/electronic-density-functional approach to materials simulations on parallel computers. In: *Computer Physics Communications* 138 (2001), Nr. 2, S. 143–154

[113] OH, K. J. ; KLEIN, M. L.: A general purpose parallel molecular dynamics simulation program. In: *Computer physics communications* 174 (2006), Nr. 7, S. 560–568

[114] OH, K. J. ; KLEIN, M. L.: A parallel molecular dynamics simulation scheme for a molecular system with bond constraints in NPT ensemble. In: *Computer physics communications* 174 (2006), Nr. 4, S. 263–269

[115] OMELYAN, I. ; MRYGLOD, I. ; FOLK, R. : Symplectic analytically integrable decomposition algorithms: classification, derivation, and application to molecular dynamics, quantum and celestial mechanics simulations. In: *Computer Physics Communications* 151 (2003), Nr. 3, S. 272–314

[116] OMELYAN, I. ; MRYGLOD, I. ; FOLK, R. : Optimized Forest–Ruth-and Suzuki-like algorithms for integration of motion in many-body systems. In: *Computer Physics Communications* 146 (2002), Nr. 2, S. 188–202

[117] O'SULLIVAN, C. : *Particulate discrete element modelling*. Taylor & Francis, 2011

[118] OWEN, D. ; FENG, Y. : Parallelised finite/discrete element simulation of multi-fracturing solids and discrete systems. In: *Engineering computations* 18 (2001), Nr. 3/4, S. 557–576

[119] PAL, A. ; AGARWALA, A. ; RAHA, S. ; BHATTACHARYA, B. : Performance metrics in a hybrid MPI–OpenMP based molecular dynamics simulation with short-range interactions. In: *Journal of Parallel and Distributed Computing* 74 (2014), Nr. 3, S. 2203–2214

[120] PANDEY, R. B. ; MILCHEV, A. ; BINDER, K. : Semidilute and concentrated polymer solutions near attractive walls: dynamic Monte Carlo simulation of density and pressure profiles of a coarse-grained model. In: *Macromolecules* 30 (1997), Nr. 4, S. 1194–1204

[121] PATRA, M. ; HYVÖNEN, M. T. ; FALCK, E. ; SABOURI-GHOMI, M. ; VATTULAINEN, I. ; KARTTUNEN, M. : Long-range interactions and parallel scalability in molecular simulations. In: *Computer physics communications* 176 (2007), Nr. 1, S. 14–22

[122] PERILLA, J. R. ; GOH, B. C. ; CASSIDY, C. K. ; LIU, B. ; BERNARDI, R. C. ; RUDACK, T. ; YU, H. ; WU, Z. ; SCHULTEN, K. : Molecular dynamics simulations of large macromolecular complexes. In: *Current opinion in structural biology* 31 (2015), S. 64–74

[123] PHILLIPS, J. C. ; BRAUN, R. ; WANG, W. ; GUMBART, J. ; TAJKHORSHID, E. ; VILLA, E. ; CHIPOT, C. ; SKEEL, R. D. ; KALE, L. ; SCHULTEN, K. : Scalable molecular dynamics with NAMD. In: *Journal of computational chemistry* 26 (2005), Nr. 16, S. 1781–1802

[124] PLIMPTON, S. : Fast parallel algorithms for short-range molecular dynamics. In: *Journal of computational physics* 117 (1995), Nr. 1, S. 1–19

[125] PLIMPTON, S. ; HEFFELFINGER, G. : Scalable parallel molecular dynamics on MIMD supercomputers. In: *Scalable High Performance Computing Conference, 1992. SHPCC-92, Proceedings.* IEEE, 1992, S. 246–251

[126] PLIMPTON, S. ; HENDRICKSON, B. : Parallel molecular dynamics simulations of organic materials. In: *International Journal of Modern Physics C* 5 (1994), Nr. 02, S. 295–298

[127] PLIMPTON, S. ; HENDRICKSON, B. : Parallel molecular dynamics algorithms for simulation of molecular systems. In: *ACS Symposium Series* Bd. 592 Citeseer, 1995, S. 114–132

[128] PLIMPTON, S. ; HENDRICKSON, B. : A new parallel method for molecular dynamics simulation of macromolecular systems. In: *Journal of Computational Chemistry* 17 (1996), Nr. 3, S. 326–337

[129] PLIMPTON, S. ; HENDRICKSON, B. ; ATTAWAY, S. ; SWEGLE, J. ; VAUGHAN, C. ; GARDNER, D. : Transient dynamics simulations: parallel algorithms for contact detection and smoothed particle hydrodynamics. In: *Proceedings of the 1996 ACM/IEEE conference on Supercomputing* IEEE Computer Society, 1996, S. 28

[130] PLIMPTON, S. J. ; SEIDEL, D. B. ; PASIK, M. F. ; COATS, R. S. ; MONTRY, G. R.: A load-balancing algorithm for a parallel electromagnetic particle-in-cell code. In: *Computer physics communications* 152 (2003), Nr. 3, S. 227–241

[131] PRIEZJEV, N. V. ; DARHUBER, A. A. ; TROIAN, S. M.: Slip behavior in liquid films on surfaces of patterned wettability: Comparison between continuum and molecular dynamics simulations. In: *Physical Review E* 71 (2005), Nr. 4, S. 041608

[132] QUANHUA SUN, I. D.: Flat plate aerodynamics at very low Reynolds number. vol. 502 (2004), S. pp. 199–206.

[133] RAPAPORT, D. : Large-scale molecular dynamics simulation using vector and parallel computers. In: *Computer physics reports* 9 (1988), Nr. 1, S. 1–53

[134] RAPAPORT, D. : Multi-million particle molecular dynamics I. Design considerations for vector processing. In: *Computer Physics Communications* 62 (1991), Nr. 2, S. 198–216

[135] RAPAPORT, D. : Multi-million particle molecular dynamics: II. Design considerations for distributed processing. In: *Computer Physics Communications* 62 (1991), Nr. 2, S. 217–228

[136] RAPAPORT, D. : Multi-million particle molecular dynamics: II. Design considerations for distributed processing. In: *Computer Physics Communications* 62 (1991), Nr. 2, S. 217–228

[137] RAPAPORT, D. : Multi-million particle molecular dynamics: II. Design considerations for distributed processing. In: *Computer Physics Communications* 62 (1991), Nr. 2, S. 217–228

[138] RAPAPORT, D. : Multi-million particle molecular dynamics: III. Design considerations for data-parallel processing. In: *Computer Physics Communications* 76 (1993), Nr. 3, S. 301–317

[139] RAPAPORT, D. : Multibillion-atom molecular dynamics simulation: Design considerations for vector-parallel processing. In: *Computer physics communications* 174 (2006), Nr. 7, S. 521–529

[140] RAPAPORT, D. C.: *The art of molecular dynamics simulation*. Cambridge university press, 2004

[141] SCHREIBER, H. ; STEINHAUSER, O. ; SCHUSTER, P. : Parallel molecular dynamics of biomolecules. In: *Parallel Computing* 18 (1992), Nr. 5, S. 557–573

[142] SCHWEITZ, J.-A. : A classical virial theorem for open systems. In: *Journal of Physics A: Mathematical and General* 10 (1977), Nr. 4, S. 507

[143] SCOTT, W. ; GUNZINGER, A. ; BÄUMLE, B. ; KOHLER, P. ; MÜLLER, U. ; MÜHLL, H. V. ; EICHENBERGER, A. ; GUGGENBÜHL, W. ; IRONMONGER, N. ; MÜLLER-PLATHE, F. u. a.: Parallel molecular dynamics on a multi signal-processor system. In: *Computer physics communications* 75 (1993), Nr. 1, S. 65–86

[144] SHAW, D. E.: A fast, scalable method for the parallel evaluation of distance-limited pairwise particle interactions. In: *Journal of computational chemistry* 26 (2005), Nr. 13, S. 1318–1328

[145] SHIMOJO, F. ; OHMURA, S. ; MOU, W. ; KALIA, R. K. ; NAKANO, A. ; VASHISHTA, P. : Large nonadiabatic quantum molecular dynamics simulations on parallel computers. In: *Computer Physics Communications* 184 (2013), Nr. 1, S. 1–8

[146] SHOJA-SANI, A. ; ROOHI, E. ; KAHROM, M. ; STEFANOV, S. : Investigation of aerodynamic characteristics of rarefied flow around NACA 0012 airfoil using DSMC and NS solvers. In: *European Journal of Mechanics-B/Fluids* 48 (2014), S. 59–74

[147] SINHA, A. B. ; SCHULTEN, K. ; HELLER, H. : Performance analysis of a parallel molecular dynamics program. In: *Computer physics communications* 78 (1994), Nr. 3, S. 265–278

[148] SMITH, L. J. ; BERENDSEN, H. J. ; GUNSTEREN, W. F.: Computer simulation of urea-water mixtures: a test of force field parameters for use in biomolecular simulation. In: *The Journal of Physical Chemistry B* 108 (2004), Nr. 3, S. 1065–1071

[149] SMITH, W. : Molecular dynamics on hypercube parallel computers. In: *Computer Physics Communications* 62 (1991), Nr. 2, S. 229–248

[150] SMITH, W. : A replicated data molecular dynamics strategy for the parallel Ewald sum. In: *Computer physics communications* 67 (1992), Nr. 3, S. 392–406

[151] SMITH, W. ; FORESTER, T. ; FINCHAM, D. : Parallel supercomputing and molecular dynamics. In: *Proc. of the 6th Joint EPS-APS International conference on physics computing*, 1994, S. 95

[152] SMITH, W. ; FORESTER, T. : Parallel macromolecular simulations and the replicated data strategy: I. The computation of atomic forces. In: *Computer physics communications* 79 (1994), Nr. 1, S. 52–62

[153] SNIR, M. : A note on n-body computations with cutoffs. In: *Theory of Computing Systems* 37 (2004), Nr. 2, S. 295–318

[154] SUITS, F. ; ELEFTHERIOU, M. ; GIAMPAPA, M. ; RAYSHUBSKIY, A. ; FITCH, B. ; PITMAN, M. C. ; WARD, T. C. ; GERMAIN, R. S.: Blue Matter on Blue Gene/L: massively parallel computation for biomolecular simulation. In: *Hardware/Software Codesign and System Synthesis, 2005. CODES+ ISSS'05. Third IEEE/ACM/IFIP International Conference on* IEEE, 2005, S. 207–212

[155] TAMAYO, P. ; MESIROV, J. P. ; BOGHOSIAN, B. M.: Parallel approaches to short range molecular dynamics simulations. In: *Proceedings of the 1991 ACM/IEEE conference on Supercomputing* ACM, 1991, S. 462–470

[156] TARMYSHOV, K. B. ; MÜLLER-PLATHE, F. : Parallelizing a molecular dynamics algorithm on a multiprocessor workstation using OpenMP. In: *Journal of chemical information and modeling* 45 (2005), Nr. 6, S. 1943–1952

[157] TARMYSHOV, K. B. ; MÜLLER-PLATHE, F. : Parallelizing a molecular dynamics algorithm on a multiprocessor workstation using OpenMP. In: *Journal of chemical information and modeling* 45 (2005), Nr. 6, S. 1943–1952

[158] TAYLOR, V. E. ; STEVENS, R. L. ; ARNOLD, K. E.: Parallel molecular dynamics: Communication requirements for massively parallel machines. In: *Frontiers of Massively Parallel Computation, 1995. Proceedings. Frontiers' 95., Fifth Symposium on the* IEEE, 1995, S. 156–163

[159] TAYLOR, V. E. ; STEVENS, R. L. ; ARNOLD, K. E.: Parallel molecular dynamics: implications for massively parallel machines. In: *Journal of Parallel and Distributed Computing* 45 (1997), Nr. 2, S. 166–175

[160] THOMPSON, P. A. ; TROIAN, S. M.: A general boundary condition for liquid flow at solid surfaces. In: *Nature* 389 (1997), Nr. 6649, S. 360–362

[161] TODD, B. ; EVANS, D. J. ; DAIVIS, P. J.: Pressure tensor for inhomogeneous fluids. In: *Physical Review E* 52 (1995), Nr. 2, S. 1627

[162] TROBEC, R. ; MERZEL, F. ; JANEZIC, D. : The complexity of parallel symplectic molecular dynamics algorithms. In: *Journal of chemical information and computer sciences* 37 (1997), Nr. 6, S. 1055–1062

[163] TUCKERMAN, M. ; BERNE, B. ; MARTYNA, G. : Reply to comment on: Reversible multiple time scale molecular dynamics. In: *The Journal of chemical physics* 99 (1993), Nr. 3, S. 2278–2279

[164] VAN DER SPOEL, D. ; LINDAHL, E. ; HESS, B. ; GROENHOF, G. ; MARK, A. E. ; BERENDSEN, H. J.: GROMACS: fast, flexible, and free. In: *Journal of computational chemistry* 26 (2005), Nr. 16, S. 1701–1718

[165] VARNIK, F. ; BASCHNAGEL, J. ; BINDER, K. : Molecular dynamics results on the pressure tensor of polymer films. In: *arXiv preprint cond-mat/0011412* (2000)

[166] VARNIK, F. ; BINDER, K. : Shear viscosity of a supercooled polymer melt via nonequilibrium molecular dynamics simulations. In: *The Journal of chemical physics* 117 (2002), Nr. 13, S. 6336

[167] VASHISHTA, P. ; NAKANO, A. ; KALIA, R. K. ; EBBSJÖ, I. : Crack propagation and fracture in ceramic films million atom molecular dynamics simulations on

parallel computers. In: *Materials Science and Engineering: B* 37 (1996), Nr. 1, S. 56–71

[168] VINCENT, J. J. ; MERZ, K. M.: A highly portable parallel implementation of AMBER4 using the message passing interface standard. In: *Journal of computational chemistry* 16 (1995), Nr. 11, S. 1420–1427

[169] WANG, L. ; LI, S. ; ZHANG, G. ; MA, Z. ; ZHANG, L. : A GPU-based parallel procedure for nonlinear analysis of complex structures using a coupled FEM/DEM approach. In: *Mathematical Problems in Engineering* 2013 (2013)

[170] WILLIAMS, J. R. ; HOLMES, D. ; TILKE, P. G.: Multi-core strategies for particle methods. (2010)

[171] WOODS, M. B. ; ALFORD, C. O.: Scaleable parallel model development and performance analysis for MIMD molecular dynamics simulations. In: *Computers & electrical engineering* 21 (1995), Nr. 3, S. 159–181

[172] WU, J.-S. ; HSU, Y.-L. ; LEE, Y.-M. : Parallel implementation of molecular dynamics simulation for short-ranged interaction. In: *Computer physics communications* 170 (2005), Nr. 2, S. 175–185